Maycon Maia Vitali

Segurança em Redes de Sensores Sem Fio: um ambiente de experimentação simulado para 6LoWPAN e um módulo de autenticação baseado em chaves pré-compartilhadas

Vitória - ES

31 de Agosto de 2012

Maycon Maia Vitali

Segurança em Redes de Sensores Sem Fio: um ambiente de experimentação simulado para 6LoWPAN e um módulo de autenticação baseado em chaves pré-compartilhadas

Dissertação apresentada para obtenção do Grau de Mestre em Informática pela Universidade Federal do Espírito Santo.

Orientador:
Prof. Dr. José Gonçalves Pereira Filho

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO - UFES
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Vitória - ES

31 de Agosto de 2012

Dissertação de Mestrado sob o título *"Segurança em Redes de Sensores Sem Fio: um ambiente de experimentação simulado para 6LoWPAN e um módulo de autenticação baseado em chaves pré-compartilhadas"*, defendida por Maycon Maia Vitali e aprovada em 31 de Agosto de 2012, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

Prof. Dr. José Gonçalves Pereira Filho
Orientador

Prof. Dr. Magnos Martinello
Universidade Federal do Rio de Janeiro

Profa. Dra. Silvana Rossetto
Universidade Federal do Rio de Janeiro

Dedicatória

À minha família, pela compreensão incalculável.

...

...

Agradecimentos

Agradeço primeiramente a Deus por todas as coisas boas que vivi, pois sei que ele só me proporcionou o melhor, Ele quem tem me proporcionado saúde e felicidade em todos os aspectos de minha vida.

Agradeço ao meu orientador José Gonçalves por ter me guiado durante toda minha jornada para o cumprimento deste trabalho e pro ter me fornecido total apoio, compreensão e dedicação, principalmente na reta final de conclusão deste trabalho.

Agrado a minha família pela minha educação, todo apoio emocional e compreensão quando quanto tive de estar ausente em mente para a conclusão deste trabalho.

Agradeço a todos os amigos que pude conhecer no LPRM (Laboratório de Pesquisa em Redes e Multimídia) pelo apoio e incentivo fornecidos em momentos que achava que não iria conseguir concluir este trabalho.

Por final, agradeço a todos aqueles que, de certa forma, colaboraram de alguma maneira para o término de desse trabalho sejam por meios técnicos, meio intelectual ou simplesmente por compreensão e incentivo. Todos esses possuem um lugar reservado eu meu coração e espero que a amizade e contato se prolongue durante diversos dias de minha vida.

Resumo

Assim como as redes de computadores convencionais, as Redes de Sensores Sem Fio (RSSF) estão sujeito aos mais diversos ataques, como negação de serviço, falsificação de endereços, comprometimento de nós, e outros ataques específicos para redes de sensores. Num cenário de emprego das RSSF em Cidades Inteligentes, a preocupação com a segurança toma uma dimensão ainda maior, considerando que dados críticos passarão a circular a todo o instante pela infra-estrutura de comunicação provida pelas redes. Neste sentido, a realização de pesquisas em segurança em RSSF se reveste de grande importância, e abre espaço para o desenvolvimento de trabalhos voltados para a criação de novos algoritmos, protocolos, ferramentas e ambientes específicos de Redes de Sensores sem Fio que contribuam para o aumento da segurança da rede.

Este trabalho propõe um ambiente de experimentação para RSSF que permite simular redes mistas, interagir com as mesmas e avaliá-las no que diz respeito ao consumo de energia, topologia estabelecida, pacotes enviados/recebidos, etc. O ambiente desenvolvido é capaz de simular redes *6LoWPAN*, sendo possível acessar os nós sensores através de seus respectivos endereços IPv6. A abordagem proposta permite que aplicações possam, remotamente, através da Internet, comunicar-se com a rede 6LoPAN, abstraindo-se do fato do ambiente ser completamente simulado. Como prova de conceito, foi simulada uma rede *6LoWPAN* e, como método de avaliação de ataques, foi implementado um nó *sybil*, que foi injetado na topologia da rede, sendo posteriormente avaliada a organização topológica da rede com e sem o nó *sybil* em execução.

Uma característica interessante da arquitetura proposta é que ela possibilita o teste remoto de aplicações; assim, uma máquina com maior capacidade de processamento pode ficar encarregada de gerar e simular a rede de sensores, enquanto as aplicações que coletam e tratam os dados são localizadas em outra máquina com menor capacidade de processamento.

Como contribuição adicional o trabalho propõe uma metologia de associação à rede que baseada em chaves pré-compartilhadas. Essa metodologia visa mitigar ataques onde nós maléficos assumem diversas identidades na rede a fim de prejudicar a visão topológica da mesma. Além disto, é apresentado um mecanismo de proteção da chave pré-compartilhada através do processo de armazenamento somente em memória volátil, o que dificulta a descoberta da chave no caso do comprometimento de um nó, ou possibilita a destruição da chave junto com a morte do sensor ao término de sua energia.

Abstract

Like traditional computer networks, Wireless Sensor Networks (WSNs) are subject to several secutity attacks, such as denial of service, spoofing addresses attack, node compromise, and other attacks specific to sensor networks. In a scenario of Smart Cities, the concerns about security takes an even greater considering that critical data will be flowing at all times by the communication paths provided by the WSN networks. In this sense, research in WSN security gains a great deal of importance, and pushes the development of research involving at the creation of new algorithms, protocols, tools and specific environments for increasing security in Wireless Sensor Networks.

This work proposes a testing environment that allows to simulate mixed WSN, a way to interact with them and evaluate the WSN concerning to energy consumption, topology, and number of packets sent and received, among other features. The testing environment is capable of simulating *6LoWPAN* networks, being able to access the sensor nodes using their IPv6 addresses. The proposed approach allows remote applications, using the Internet, to communicate with the 6LoWPAN network, apart from the fact that the environment is fully simulated. As proof of concept, we simulated a *6LoWPAN* network, promoting an *sybil* attack in the network topology, and subsequently evaluated the organization of the network topology with and without the *sybil* node running.

An interesting feature of the proposed architecture is that it enables remote testing of applications, so a machine with more processing capacity may be responsible for generating and simulating the sensor network, while applications that collect and process data are located in another machine with less processing power.

As an additional contribution, we propose a new methodology for network node association based on pre-shared keys. The methodology aims to mitigate attacks where malicious nodes assume various identities in the network in order to impair the vision of the same topological. Furthermore, we present a protection mechanism by storing the pre-shared key only in volatile memory, which makes more difficult the discovery of the key in the case of a node failure, or even the destruction of the key together with the death of the sensor when your energy comes to the end.

Sumário

Lista de Figuras

Lista de Siglas e Abreviaturas

MEMS	Microelectromechanical systems
RSSF	Redes de Sensores Sem Fio
IoT	Internet of Things
DoS	Denial of Service
IPv6	*Internet Protocol* (versão 6)
6LoWPAN	*IPv6 over Low power Wireless Personal Area Networks*
IETF	Internet Engineering Task Force
IEEE	Institute of Electrical and Electronics Engineers
ICMP	Internet control Message Protocol
UDP	user Datagram Protocol
TCP	Transmission Control Protocol

1 Introdução

1.1 Motivação

O avanço acelerado nas tecnologias de microeletrônica (MEMS), da comunicação sem fio e da eletrônica digital tem viabilizado a construção de dispositivos sensores miniaturizados de baixo custo e de baixo consumo, dotados de capacidade básicas de sensoreamento, processamento, armazenamento e comunicação sem fio. Esses dispositivos podem ser interligados segundo um arranjo topológico qualquer, por exemplo, em malha ou estrela, formando uma poderosa infra-estrutura de coleta e disseminação de dados, comumente chamada de Rede de Sensores sem Fio (LOUREIRO et al., 2003).

As Redes de Sensores Sem Fio (RSSF) prometem revolucionar a atividade de sensoriamento em vista do seu tamanho, baixo custo, flexibilidade, precisão e facilidade de implantação. Nós sensores podem ser implantados em locais inóspitos e de difícil acesso humano, e servir como fonte de informação para a tomada de decisão em diferentes cenários de aplicação. O potencial de observação e controle do mundo real permite que essas redes miniaturizadas se apresentem como uma solução para a implantação de diversos serviços e aplicações em cenários como monitoramento ambiental, gerenciamento de infra-estrutura de datacenters, aplicações militares e de segurança pública, monitoramento e controle industrial, composição de solos e monitoramento de pragas na agricultura, dentre várias outras. A visão é que as RSSF se tornem disponíveis em todos os lugares, executando as tarefas das mais diversas possíveis, num cenário típico de computação ubíqua.

Este potencial tem estimulado o desenvolvimento de hardware e software para RSSF e tem

atraído a atenção da comunidade acadêmica e da indústria. Recentemente, as RSSF vem sendo explorados na viabilização do conceito de Cidades Inteligentes (VASSEUR; DUNKELS, 2010) e também como uma das bases para o que vem sendo denominado de Internet das Coisas (*IoT - Internet of Things*). Cidade Inteligente, por exemplo, é um conceito que está relacionado à oferta de serviços de informação e comunicação para solucionar problemas dos cidadãos em seus conglomerados urbanos, tais como trânsito, vigilância patrimonial, atendimento de emergência, monitoramento ambiental, saúde, educação e inclusão digital, além de potencialmente permitir uma maior eficiência e transparência na gestão pública, devido ao volume, cruzamento e maior precisão dos dados coletados.

Para suportar a implementação de soluções para esses problemas, dados urbanos precisam ser coletados e disseminados através de infra-estruturas de comunicação que, por sua vez, exigem formas integradas, heterogêneas, e inteligentes de comunicação. Neste sentido, as Redes de Sensores sem Fio se mostram extremamente adequadas para este fim.

Observa-se, entretanto, que as RSSF operam sob uma forte restrição de energia, o que torna desafiador projetar uma rede que se mantenha viva o máximo de tempo possível (POTDAR; SHARIF; CHANG, 2009). Além disso, devido a essa restrição, protocolos desenvolvidos para redes convencionais, por exemplo, protocolos das camadas de enlace e de rede, não se adaptam diretamente às redes de sensores sem fio. Portanto, estudos para adaptação e criação de novos protocolos tem sido necessários e realizados ao longo dos anos, com o objetivo de se definir uma arquitetura de comunicação robusta, que leve em consideração as limitações dos dispositivos.

Não muito diferente de uma rede convencional, as Redes de Sensores sem Fio também possuem problemas e desafios na área de segurança e necessitam tratar esses requisitos no contexto de segurança da informação, particularmente em um cenário de Cidades Inteligentes. É sabido que, quando se fala em segurança da informação, alguns pilares devem ser mantidos: a *disponibilidade*, a *integridade*, a *autenticidade* e a *confidencialidade*. O mesmo não é diferente no escopo das Redes de Sensores Sem Fio, sendo necessário a implementação de mecanismos que protejam a rede estabelecida tendo por base esses quatro principais aspectos. Resumidamente, *(i)* a disponibilidade visa manter os nós acessíveis sempre que necessário, pois os mesmos po-

dem, por exemplo, sofrer ataques de Negação de Serviço[1]; *(ii)* a integridade de uma rede visa garantir que qualquer nó ou dado transmitido não seja alterado; *(iii)* a autenticidade visa garantir que um determinado nó sempre tenha uma identidade verdadeira, sendo, desta forma, autêntico e confiável para os demais nós da rede; e *(iv)* a confidencialidade objetiva fornecer um meio de comunicação onde um nó não autorizado não consiga ter acesso às informações sigilosas.

Dentre os pilares descritos, a autenticidade pode ser ferida por diversos ataques; assim, um dos objetivos de se introduzir mecanismos de autenticidade em RSSF é garantir a confiança entre os nós. Garante-se, dessa de forma, que seja possível identificar se um determinado nó é realmente autêntico e se ele tem mesmo permissão para fornecer ou coletar informações da rede. Como exemplos de ataques efetuados a fim de quebrar a autenticidade da rede, estão nós maléficos assumirem identidades legítimas para a rede falsificando seu endereço original com o intuito dez fornecer dados de sensoreamento inválidos; ou até mesmo nós serem adicionados à rede a fim de injetar dados incertos que influenciem nas diversas tomadas de decisões.

A maioria das abordagens de defesa contra ataques de autenticidade disponibilizadas na literatura baseiam-se em criptografia de dados(MUNIVEL; AJIT, 2010; RODRIGUEZ et al., 2008; XU; GE, 2009). Entretanto, em geral, as estratégias de comunicação de dados cifrados, transferência de chaves e processamento criptográfico, apresentam um considerável consumo de energia, o que deve ser evitado nas RSSF. A garantia da autenticidade pode ser obtida com a existência de modelos ou ambientes de autenticação dotados de mecanismos que permitam garantir que um determinado nó é legítimo na rede atual. A proposição de um ambiente que proveja tal autenticação constitui uma das contribuições deste trabalho. Observa-se que, no escopo do trabalho realizado nesta dissertação, é somente o pilar da autenticidade que é investigado e tratado na solução de segurança proposta.

Normalmente, com o intuito de mensurar e comparar propostas e implementações de segurança, são feitos testes experimentais utilizando-se duas abordagens principais: ambientes de experimentação real, provido pelos (*testbeds*); ou ambientes de experimentação simulados. Os ambientes simulados e os *testbeds* são fundamentais para análise do impacto das possíveis

[1]Do inglês, *DoS - Denial of Service*

ameaças de uma Redes de Sensores Sem Fio, pois torna possível avaliar o quão resistente está a rede, os impactos de mudanças de topologia, a adequação de protocolos, verificar os limites de segurança de uma rede, e auxiliar na avaliação da efetividade de possíveis contra-medidas.

Os *testbeds* consistem em prover ambientes de experimentação real. Geralmente, possuem uma infra-estrutura com uma diversidade de *hardware* e uma interface onde é possível gravar o *software* nos nós sensores e coletar os resultados. Os ambientes de experimentação simulados são utilizados muitas vezes quando os obstáculos de se obter os equipamentos necessários para um ambiente real e, mesmo quando presentes, a dificuldade de se conseguir os resultados de um experimento de maneira simples e produtiva, impossibilitam a utilização de ambientes reais. Além disto, os ambientes simulados fornecem algumas vantagens em comparação com os ambientes reais, como a facilidade na coleta de grandezas (consumo de energia, por exemplo), uma interface de análise e depuração da aplicação[2] etc. Cada simulador possui as suas vantagens e desvantagens. Alguns não permitem a simulação de imagens de aplicações distintas (LEVIS; LEE; WELSH, 2003), outros não suportam nativamente plataformas de redes de sensores (NS-2..., 2012) ou, quanto suportam, limitam-se a um único tipo de plataformas (LEVIS; LEE; WELSH, 2003; AVRORA..., 2012a).

Assim, um ambiente de simulação fornece um meio de testar as aplicações e obter seus resultados de maneira simples, eficaz e, principalmente, barata. Diferente de um *testbed*, onde existe a limitação física de quantidade de nós sensores, em ambientes simulados esta limitação é diretamente proporcional à capacidade de processamento e memória (RAM) da máquina.

Ao longo dos estudos realizados, verificou-se que, apesar de existirem diversos ambientes de simulação para RSSF, nenhum deles - pelo menos do nosso conhecimento - oferece meios de se simular, de maneira unívoca a um *testbed*, uma rede IPv6 de dispositivos de baixo consumo, mais precisamente, uma implementação do padrão 6LoWPAN. O 6LoWPAN, descrito em mais detalhes no próximo capítulo, é um padrão do IETF cujo esforços são especialmente direcionados para a criação de uma solução de uma arquitetura de integração de redes de sensores com a Internet, que se apoia na transmissão de pacotes IPv6 sobre enlaces IEEE 802.15.4. Modelos

[2] *Debugging*

de ataques sofisticados podem surgir contra aplicações que utilizem tais tecnologias, tornando crucial o desenvolvimento de ambientes para se testar os ataques contra redes 6LoWPAN.

Durante a avaliação dos simuladores disponíveis, verificou-se que nenhum deles suportava plenamente a tarefa de realização de testes de simulação de redes 6LoWPAN o que, consequentemente, inviabilizava a criação de ambientes de testes de segurança sobre essas redes. O simulador *Avrora* (AVRORA. . . , 2012a), adotado no trabalho, apesar de simular redes mistas e permitir a fácil coleta dos resultados, não fornece um meio de comunicação direto com os nós sensores através do IPv6. Além disso, o *ip-driver*, responsável por criar tal comunicação, não consegue interagir diretamente com o simulador *Avrora*. Deste modo, há uma necessidade imediata de se desenvolver uma arquitetura de testes para redes 6LoWPAN onde seja possível criar um túnel de comunicação entre o Avrora e o *ip-driver*, como forma a inibir as limitações impostas.

Além disso, é desejável que a arquitetura possibilite o teste remoto de aplicações, permitindo que uma máquina com maior capacidade de processamento seja responsável por gerar e simular a rede, enquanto as aplicações que coletam e tratam os dados possam estar localizadas em outra máquina com menor capacidade de processamento. Tal solução constitui uma contribuição importante desta dissertação.

1.2 Objetivos

O trabalho visa conceber um ambiente de experimentação para Redes de Sensores Sem Fio que permita simular redes 6LoWPAN mistas, interagir com as mesmas e avaliá-las no que diz respeito ao consumo de memória e energia. Tal ambiente pode ser usado como base para o desenvolvimento de testes de segurança em redes de sensores sem fio. Como forma de avaliar a funcionalidade do ambiente proposto, o ataque *Sybil* é simulado com o intuito de levantar os impactos causados na rede.

Particularmente em relação ao pilar da autenticidade, o trabalho propõe uma metologia de associação que, baseada em chaves pré-compartilhadas, visa mitigar tais ataques, em que nós

maléficos assumem diversas identidades na rede a fim de prejudicar a sua visão topológica. Além disto, é apresentado um mecanismo de proteção da chave pré-compartilhada através do processo de armazenamento somente em memória volátil, o que dificultaria a descoberta da chave no caso do comprometimento de um nó ou possibilitaria a destruição da chave em caso de término da bateria.

O trabalhou foi desenvolvido para sistema operacional TinyOS, usando a plataforma de nós sensores *MicaZ* que, apesar de possuir poucos recursos, foi suficiente para atender à proposta do trabalho. Como ambiente de simulação, foi empregado o Avrora e algumas ferramentas auxiliares necessárias para a perfeita simulação do ambiente proposto.

1.3 Organização

O Capítulo 2 apresenta o conjunto de tecnologias investigadas durante o desenvolvimento do trabalho, com destaque para o sistema operacional TinyOS e a sua linguagem associada nesC. Neste capítulo também é realizada uma breve introdução ao padrão IEEE 802.15.4 e ao padrão 6LoWPAN, bem como aos simuladores de RSSF, em especial o Avrora. O Capítulo 3 apresenta as principais ameaças que uma Sede de Sensores sem Fio pode estar sujeita, cobrindo os ataques mais comuns que podem acometer essas redes. O Capítulo 4 introduz o ambiente de simulação proposto, utilizado como modelo de experimentação. O capítulo apresenta um exemplo de ataque de autenticidade, onde um nó adentra à rede e efetua um ataque *Sybil*. O Capítulo 5 apresenta o modelo de associação proposto para Redes de Sensores Sem Fio, que visa mitigar ataques que utilizam a injeção de nós maléficos à rede. Neste mesmo capítulo são apresentados avaliações do consumo de memória e consumo de energia para uma rede com e sem a proposta. O Capítulo 6 encerra o trabalho apresento as considerações finais e as expectativas de trabalhos futuros.

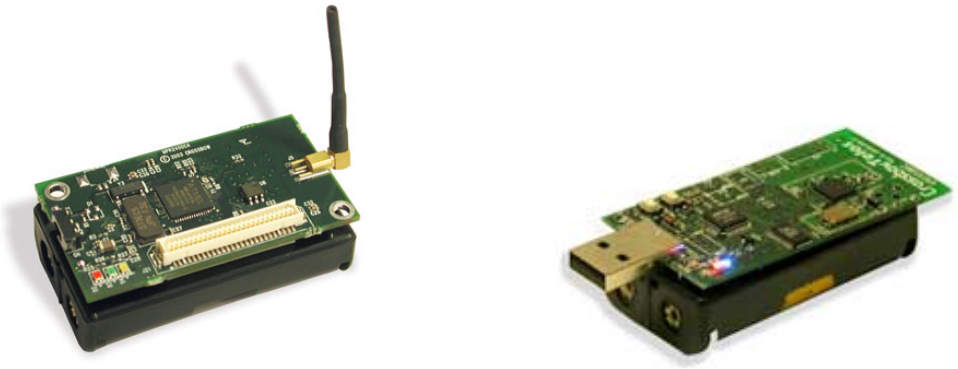
2 *Referencial Teórico*

2.1 Introdução

Neste capítulo são apresentadas as principais tecnologias e referenciais teóricos usados no desenvolvimento do trabalho. Inicialmente, na Seção 2.2, é feita uma breve introdução às Redes de Sensores Sem Fio, abordando suas principais características, desafios e alguns exemplos de aplicações. A Seção 2.3 introduz a plataforma de desenvolvimento TinyOS e sua linguagem de desenvolvimento nesC. TinyOS e nesC são, respectivamente, o sistema operacional e a linguagem de programação mais difundidas atualmente(DARGIE; POELLABAUER, 2010), sendo ambos adotados no projeto $(CIA)^2$ e nesta dissertação. A Seção 2.4 é apresentado o padrão IEEE 802.15.4 como a base de diversas pilhas de protocolos para comunicação em redes de sensores sem fio e dispositivos embarcados. A Seção 2.5 apresenta o 6LoWPAN, padrão IETF para integração de dispositivos de baixo consumo do tipo IEEE 802.15.4 com a Internet, baseado em uma pilha IPv6. A Seção 2.6 introduz uma implementação do padrão 6LoWPAN para TinyOS, denominada *blip*. A Seção 2.7 apresenta exemplos de simuladores usados em RSSF e, em especial, introduz e justifica o uso o simulador Avrora no trabalho. A Seção 2.10 conclui o capítulo.

2.2 Redes de Sensores Sem Fio

Os avanços recentes na tecnologia micro-eletromecânica (MEMS), da comunicação sem fio e da eletrônica digital tem permitido a construção de nós sensores de baixo-custo, baixo-consumo e multifuncionais, pequenos em tamanho e não limitados a pequenas distâncias (LOU-



(a) Nó sensor MICAz (b) Nós senstor TelosB

Figura 2.1: Exemplo de nós sensores

REIRO et al., 2003). Com a diminuição dos custos e maior autonomia de energia, aliada a uma capacidade básica de processamento, armazenamento e comunicação, tornou-se possível a criação de uma gama de novas aplicações de monitoramento de maior precisão em diversas áreas, inclusive em ambientes completamente hostis, baseadas no uso de um número elevado de sensores sem fio. As Figuras 2.1a e 2.1b, ilustram dois exemplos de plataforma de nós sensores, ambos desenvolvidos pela empresa *Crossbow Technology*[1].

Uma Rede de Sensores Sem Fio (RSSF) é composta por uma grande quantidade desses nós sensores, que são espalhados dentro ou muito próximo a área, objeto ou fenômeno a ser monitorado(LOUREIRO et al., 2003). Tais fenômenos podem estar relacionados às correntes marítimas, animais, estruturas civis ou mesmo regiões inóspitas, como vulcões; logo, é possível desenvolver aplicações baseadas em RSSF para acompanhar e gerar relatórios de mudanças marítimas, para relatar a posição de animais – podendo apresentar a sua trajetória e entender como funciona o processo de migração de uma determinada espécie – ou para monitorar a variação da temperatura e detecção de abalos físicos a fim de prever uma possível erupção de um vulcão (Figura 2.2). Exemplos de aplicações de RSSF incluem: monitoramento ambiental e de habitat em cidades, rios, mares, matas e florestas, rastreamento de objetos, pessoas e animais, controle de segurança física, aplicações de *health* e *homecare*, monitoramento de estruturas civis, monitoramento de facilidades(*facilities*[2]) em *datacenters*, automação industrial, aplicações

[1] http://www.xbow.com

[2] Como sistemas de HVAC, controle de incêndios, sistemas de geradores, etc

militares, e uma enorme quantidade de outras aplicações de monitoramento de grandezas físicas, como temperatura, pressão, umidade, movimento veicular, luminosidade, níveis de ruído, composição do solo, níveis de estresse mecânicos, etc.

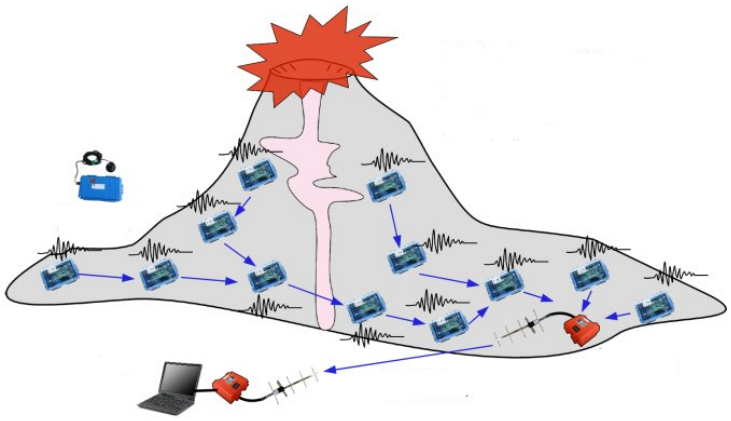


Figura 2.2: Exemplo de aplicação de RSSF em monitoramento de abalos sísmicos

Como os nós sensores possuem baixo custo e, dependendo da aplicação, sua manutenção pode ser considerada inviável, um dos grandes desafios em RSSF consiste em aumentar seu tempo de vida diminuindo o seu consumo de energia, por esta ser a forma natural de "morte" de um sensor. Desta maneira, praticamente todas as propostas de metodologias e abordagens aplicadas a RSSF, por exemplo, projetos de protocolos de camada de enlace e de rede, levam em consideração a restrição extrema de energia. Por conta desta restrição, os protocolos de redes tradicionais não se adaptam às RSSF; portanto, pesquisas e propostas de adaptação e desenvolvimento de novos protocolos específicos para as RSSF tem sido feitos ao longo dos anos, nas diversas camadas da arquitetura de redes.

A arquitetura de um nó sensor consiste basicamente de uma unidade microcontroladora (UMC) que interconecta um transceptor de dados, sensores de grandezas físicas, juntamente com um conversor analógico para digital, uma fonte de energia e uma memória *flash* externa. A UMC inclui um processador, uma memória RAM (SRAM) e uma memória *flash* interna. A pouca energia dos microcontroladores limita o armazenamento, normalmente menos de 10 KB de SRAM, do código de execução. A quantidade limitada de memória *flash* no *chip* possibilita apenas uma pequena área de memória não volátil; logo, uma grande quantidade de memória flash extra é usada em uma placa separada para possibilitar a melhor funcionalidade da rede

(RIZVI; CHUNG, 2008). Exemplos de plataformas incluem: MICAz, TelosB e Iris. A plataforma MICAz, por exemplo, possui a menor quantidade de memória RAM, 4kB, se comparada às plataformas citadas. Ela é dotada de um rádio de baixo alcance e antena para conexão sem fio, 512kB de memória flash, e conector de expansão que permite que sejam conectadas placas de sensores de som, luminosidade, temperatura e também serve para que o nó seja conectado à estação base (*base station*[3]). A plataforma TelosB possui 10 kB de memória RAM e a IRIS 8 kB de RAM.

Figura 2.3: Exemplo de um nó sensor de uma RSSF

2.2.1 Estabelecimento de uma RSSF

A Figura 2.4 ilustra um exemplo de como pode ser estabelecida uma Rede de Sensores Sem Fio. Na figura, os dispositivos sensores são inicialmente lançados no ambiente a ser monitorado, que pode, eventualmente, ser de difícil acesso ou mesmo hostil. Em seguida a rede se auto-organiza, (Seção 2.2.2), de forma que gere as tabelas de roteamento e os sensores passem a trabalhar buscando o menor consumo de energia possível.

2.2.2 Auto-Organização

A auto-organização[4] é uma característica essencial para o estabelecimento de uma Rede de Sensores Sem Fio. Sabe-se que o consumo de energia de um transceptor é diretamente proporcional ao tempo que ele fica ligado, à quantidade de informações enviadas, e à distância em que as informações foram enviadas. Em uma Rede de Sensores Sem Fio normalmente os sensores não possuem posição fixa e estão sujeitos à destruição ou ao término de sua bateria,

[3]Estação que permite com que os dados da rede sejam acessados por um computador e vice-versa através do conector USB

[4]Do inglês *Self-organization*

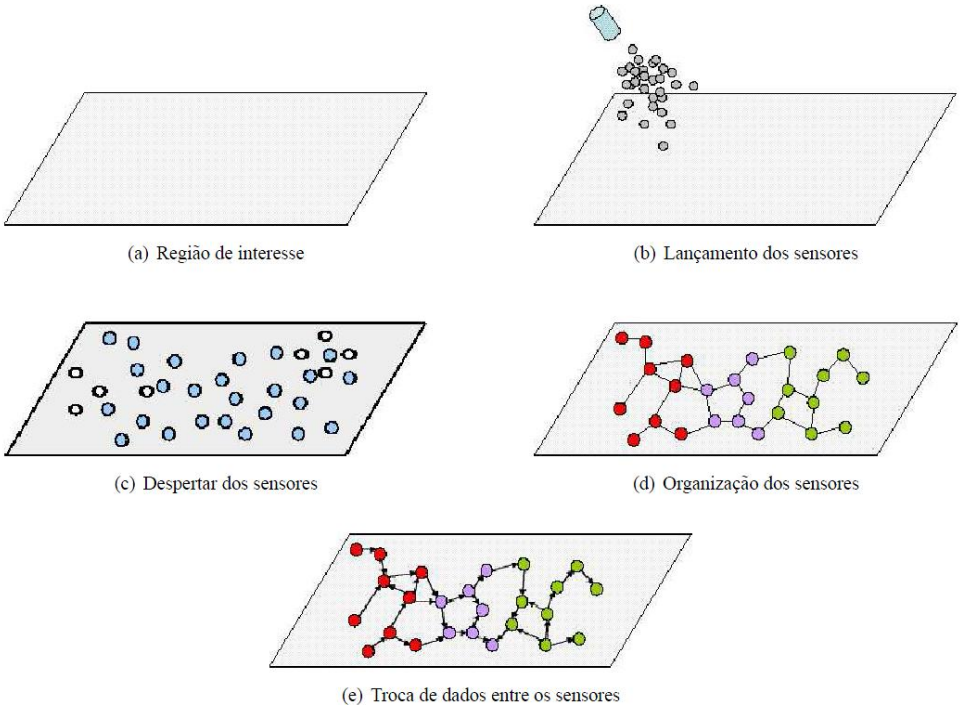


Figura 2.4: Estabelecimento de uma rede de sensores sem fio(LOUREIRO et al., 2003)

tornando a topologia da rede dinâmica. Com isso, ela deve ser capaz de se auto-organizar de forma a reduzir o consumo de energia, modificando as tabelas de roteamento e, consequentemente, reduzindo a distância em que terão que enviar as informações. A auto-organização é, portanto, uma das características mais fundamentais das redes de sensores sem fio, permitindo a pronta adaptação às mudanças topológicas que ocorrem no seu ciclo de vida devido à morte e à entrada de nós. Por outro lado, do ponto de vista da segurança, a auto-configuração pode se constituir um problema, podendo ser um ponto de entrada de nós maliciosos que exploram falhas de controle de autenticidade de nós ingressantes na rede.

2.2.3 Auto-Modificação

A maioria dos nós sensores possui arquitetura Harvard, ou seja, a gravação do software no sensor é feita em tempo de desenvolvimento e, após a implantação da rede, não é possível mais ter modificação. Isso se deve à característica da arquitetura de ter a memória de dados separada da memória de programa, o que dificulta a alteração do software que está implantado no sensor. Porém, alguns microprocessadores, como o Atmega128, utilizado nos nós sensores MicaZ, possuem instruções que copiam informações da memória de dados para a memória

de programa, o que torna possível implementar um mecanismo de auto-modificação nos nós sensores.

A auto-modificação[5] é utilizada quando se deseja corrigir *bugs*, implementar novas funcionalidades, ou até mesmo modificar completamente algumas funcionalidades do sensor, como o algoritmo de roteamento empregado.

2.3 TinyOS e a Linguagem NesC

2.3.1 Introdução

TinyOS (TINYOS..., 2012) é um sistema operacional baseado em eventos desenvolvido para dispositivos de baixo consumo. O TinyOS é o sistema operacional mais usado, documentado e com o maior número de ferramentas de desenvolvimento em ambiente de RSSF. Tendo passado por um longo período de projeto e apresenta um contínuo processo de evolução, o que o torna o sistema operacional para RSSF mais estável e difundido atualmente, adequado para vários tipos de aplicações. Conceitualmente, sua arquitetura é compacta, sendo constituída de um escalonador e um conjunto de componentes, os quais podem ser ligados através de interfaces bem definidas. Os componentes são classificados em componentes de configuração e módulos. Um componente de configuração especifica como dois ou mais módulos estão conectados (o que é denominado de *wiring*) enquanto que os módulos formam os blocos básicos de um programa TinyOS, isto é, os módulos representam as implementações. Em outras palavras, a configuração liga diversos componentes e os módulos implementam suas funções. A composição de múltiplas configurações em um único código executável produz uma aplicação TinyOS. Por construção, o TinyOS não provê uma separação clara entre S.O. e a aplicação. As figuras 2.5 a 2.7 ilustram a estrutura lógica de componentes e configurações.

Na Figura 2.5 o componente A declara o seu serviço oferecendo a interface C que, por sua vez, provê o comando D1 e sinaliza o evento D2. Na Figura 2.6, o componente B expressa interesse na interface C declarando uma chamada ao comando D1 e provendo uma rotina para

[5] *Self-updating* ou *Self-modification* em inglês

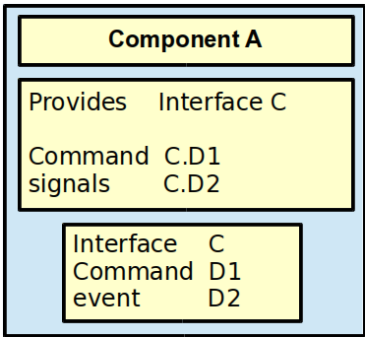


Figura 2.5: Um componente TinyOS que provê uma interface

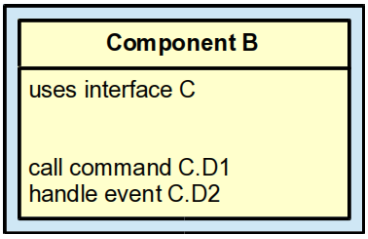


Figura 2.6: Um componente TinyOS que provê uma interface

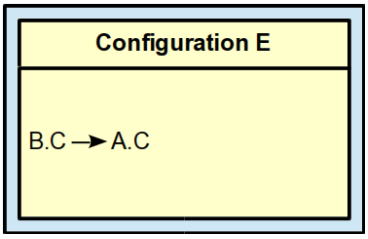


Figura 2.7: Um componente TinyOS que liga dois compoenentes

processar o evento D2. Na Figura 2.7, a ligação entre os componentes A e B pela interface C é estabelecida através da configuração E.

Os programas e aplicações TinyOS são escritos na linguagem nesC (GAY et al., 2003; LEVIS, 2006). NesC é uma extensão da linguagem de programação C, e foi projetada para desenvolvimento em RSSF. A linguagem fornece um modelo de execução baseado em eventos, um modelo de concorrência flexível e uma interface de projeto de aplicações orientadas a componentes. Um componente encapsula um estado e interage através de interfaces bem definidas. Uma interface pode definir comandos (*commands*), tratadores de eventos (*event handlers*) e atividades (*tasks*). Cada componente declara formalmente os comandos que usa e os eventos que sinaliza. Desta maneira, é possível determinar em tempo de compilação os recursos requeridos pela aplicação.

Os componentes são estruturados hierquicamente e comunicam-se entre si através de comandos e eventos: componentes hierarquicamente superiores emitem comandos para os componentes mais baixos na hierarquia, enquanto que esses sinalizam eventos para os componentes de mais alto nível. Desta forma, componentes hierarquicamente superiores implementam tratadores de eventos e os componentes inferiores implementam os processadores de comandos (ou as funções de sub-rotina).

Comandos, atividades e eventos são os blocos fundamentais do ambiente de execução do TinyOS, sendo os responsáveis efetivos pela comunicação entre os componentes. Os comandos são as funções propriamente ditas, implementadas pelos módulos que a aplicação utiliza. Eles são acionados através da palavra reservada *call* seguido da função que se deseja executar em algum componente específico. Por exemplo, para se ascender o *led0* de uma interface *Leds* deve-se utilizar a seguinte chamada na linguagem Nesc: **`call`** `Leds.led0On();`. Durante a construção de um componente, os comandos devem ser rotinas que possuem a palavra reservada **`command`** como prefixo em sua declaração. No TinyOS, os comandos são sempre operações não bloqueantes. Para lidar com operações de longo tempo de execução, o TinyOS introduz o conceito de operação *split-phase*. Nessa, uma chamada de função retorna imediatamente e a função chamadora notifica através de um evento o componente chamador quando a operação for finalizada.

As atividades (*tasks*) são rotinas executadas pelo escalonador de tarefas do TinyOS. Essas rotinas são declaradas com o prefixo **`task`** e não devem conter parâmetros e retorno. Uma tarefa é enviada para a fila de execução do escalonador de tarefas através da chamada **`post`** `nomeDaTarefa`. As tarefas são processos monolíticos que devem ser executados até o seu término, isto é, elas são não-preemptivas. Elas não podem ser "preemptadas" por outras tarefas, embora possam ser interrompidas por eventos. É desta forma que o TinyOS suportam concorrência e garante que tarefas não interferem em outras e não corrompem dados de outras tarefas. O escalonador de tarefas do TinyOS adota a política de escalonamento FIFO.

Cada componente interessado em um evento particular deve prover um tratador de evento para processá-lo. Tratadores de eventos (*event handlers*) serão chamados quando o evento es-

perado ocorrer. Componentes de mais baixo nível possuem tratadores de eventos que estão diretamente ligados a interrupções de *hardware*, tais como interrupções de relógio ou geradores de eventos. Um tratador de evento pode reagir a um evento de diferentes maneiras, por exemplo, sinalizando eventos de mais alto nível, chamando comandos de mais baixo nível, depositando uma informação no seu contexto de memória ou postando uma tarefa, etc.

Por fim, observa-se que a alocação de memória no TinyOS é feita de maneira estática, com o objetivo específico de otimizar recursos. Como os requisitos de memória da aplicação são conhecidos a priori, em tempo de sua composição, evita-se o (*overhead*) extra associado à alocação dinâmica.

2.4 O Padrão IEEE 802.15.4

O padrão IEEE 802.15.4 (SOCIETY, 2003) é a base de diversas pilhas de protocolos para comunicação em redes de sensores sem fio e dispositivos embarcados. O padrão IEEE 802.15.4 define um conjunto de funções e protocolos das camadas física e enlace da arquitetura de rede, sendo usado em soluções integradas, como o Zigbee Alliance(ZIGBEE...,) e WirelessHART(HART...,), bem como em soluções *open-source*, como é o caso dos sistemas operacionais TinyOS e o Contiki(CONTIKI...,).

O padrão é amplamente adotado em sistemas e aplicações embarcadas para prover vários pontos de sensoreamento e controle com baixo custo. Essas aplicações tipicamente requerem uma grande quantidade de nós, os quais se comunicam através de um ou vários saltos com a finalidade de cobrir uma determinada área geográfica de interesse. Os nós da rede devem ser capazes de operar em regiões desprotegidas e por um longo período de tempo, sustentados por baterias ou pilhas (fontes de energia esgotáveis). Como os objetivos principais do padrão IEEE 802.15.4 são a economia de recursos e a simplicidade de programação, ele torna-se bastante atraente para a comunicação sem fio entre dispositivos com altas restrições de recursos computacionais e energéticos, como é o caso das redes de sensores sem fio.

Entre as características do padrão IEEE 802.15.4, destacam-se:

- Baixa taxa de transmissão: a camada física pode ser configurada para operar em duas faixas de frequência - 868/915MHz (taxas de 20kbps, 40kbps e opcionalmente 100kbps e 250Kbps), ou 2.4GHz (taxas de 250Kbps);

- Baixa potência: os dispositivos são sustentados por baterias com perspectiva de operação durante meses ou anos;

- Baixo custo: normalmente o rádio é acoplado a dispositivos embutidos com recursos de hardware limitados;

- Distâncias curtas: o alcance do sinal varia de 10m a 100m;

- Diferentes tipos de dispositivos: os dispositivos podem ser do tipo FFD (Full Function Device) – podem ser coordenadores de grupos e roteadores de pacotes – ou RFD (Reduced Function Device) – dispositivos com funções limitadas (permanecem em modo inativo a maior parte do tempo);

- Diferentes modos de operação: dois modos de transmissão são definidos para a camada MAC – beacon-enabled e nonbeacon-enabled (no modo nonbeacon-enabled usa-se o mecanismo CSMA-CA para evitar colisões);

- Reserva de slots de tempo: permite garantir requisitos de tempo real;

- Segurança: suporte integrado para comunicações seguras na camada de enlace (algoritmo de criptografia AES2).

Os quadros IEEE 802.15.4 são de quatro tipos: sinalização (*beacon*), comandos MAC, *acks* e dados. Os datagramas da camada de rede (IPv6 ou IPv4) devem ser carregados em quadros de dados. Os quadros de dados podem opcionalmente requisitar confirmação de recebimento (emphacks).

O padrão IEEE 802.15.4 permite endereçamento de 64 bits e endereçamento curto de 16 bits para regiões da rede. Além disso, possui endereçamento *broadcast* e *unicast*; porém, não admite endereçamento *multicast*. Como os dispositivos IEEE 802.15.4 possuem endereçamentos de 16

e 64 bits, é necessário ter uma forma única de identificar uma interface, no caso de pilhas de protocolos que admitem endereçamento IPv6, como é o caso do 6LoWPAN (vide a seguir).

2.5 O Padrão 6LoWPAN

Nos anos recentes, particularmente com o surgimento das pesquisas em Internet das Coisas (*IoT - Internet of Things*) e com o movimento em direção às Cidades Inteligentes, presenciamos o surgimento de vários esforços visando à integração das redes de dispositivos de baixo consumo, como as redes de sensores sem fio, com a Internet.

As vantagens provenientes da integração de dispositivos de baixo consumo baseados no padrão IEEE 802.15.4 com a Internet são evidentes, e podem ser vistas a partir de diferentes ângulos. Sob o ponto de vista das aplicações que usam esses dispositivos, por exemplo, o uso de padrões e protocolos bem conhecidos e testados da Internet facilita a difusão dos dados coletados e o uso combinado desse dados com informações provenientes de outras aplicações. Sob o ponto de vista da Internet, a implementação da pilha de protocolos IP em dispositivos com capacidades limitadas possibilita estender o seu alcance para além dos computadores, deixando de ser uma rede que conecta computadores para se tornar uma rede que também conecta dispositivos.

De particular interesse para este trabalho é o modelo de referência proposto pelo grupo de trabalho do IETF, denominado de *6LoWPAN* para integração de dispositivos IEEE 802.15.4 o com a Internet tendo por base o suporte a IPv6 (KUSHALNAGAR; SCHUMACHER; MONTENEGRO, 2007) e uma implementação deste modelo disponibilizada para o ambiente de programação TinyOS, denominada de *blip* (TINYOS-BLIP,).

6LoWPAN é um acrônimo para (*IPv6 over Low power Wireless Personal Area Networks*). É um grupo de trabalho do IETF responsável pela definição de normas e protocolos que permitem o tráfego de pacotes com endereços IPv6 sobre redes IEEE 802.15.4. Observe que isso permite endereçar dispositivos inteligentes diversos e os nós das redes de sensores sem fio usando o espaço de endereços ampliado do IPv6, facilitando a implantação de soluções em cenários de

Internet das Coisas e Cidades Inteligentes.

Atualmente, a implementação de redes de dispositivos de baixo consumo é efetuada por um conjunto de tecnologias proprietárias o que dificulta a interoperabilidade entre diferentes redes e a sua inserção em serviços baseados na Internet. A integração da pilha IP a essas redes permite superar este problema e introduz um conjunto de vantagens, nomeadamente (KUSHALNAGAR; SCHUMACHER; MONTENEGRO, 2007):

- a natureza ubíqua das redes IP permite a utilização de infra-estruturas já existentes;
- a tecnologia IP é bem conhecida e já se encontra devidamente testada;
- o protocolo IP é definido em especificações do IETF, que são disponibilizadas livremente;
- já existem um conjunto de ferramentas disponíveis para diagnóstico e gestão de redes IP;
- dispositivos com conectividade IP podem ser ligados a uma rede IP sem necessidade de um gateway ou proxy.

No entanto, o hardware das redes IEEE 802.15.4 tem várias limitações, que dificultam a implementação do protocolo IP. Nomeadamente, o tamanho máximo dos quadros é de apenas 127 bytes, o alcance de transmissão é no máximo de algumas dezenas de metros, a taxa de transmissão máxima é de 250 kbps e os recursos de memória são limitados. Por isso, a adaptação do protocolo IP a uma rede IEEE 802.15.4 levanta um conjunto de problemas, incluindo (KUSHALNAGAR; SCHUMACHER; MONTENEGRO, 2007; MONTENEGRO et al., 2007):

- um pacote IPv6 deve sempre que possível ser enviado em apenas um quadro IEEE 802.15.4, de forma a evitar uma fragmentação excessiva;
- as soluções atuais para encaminhamento IP podem não ser adequadas para a topologia (*mesh*), que é a topologia normalmente utilizada;
- o IPv6 inclui mecanismos de segurança (IPsec) que podem ser demasiado complexos para serem usados em dispositivos IEEE 802.15.4;

- os protocolos da pilha TCP/IP não estão otimizados para funcionar em redes LR-WPAN;
- os nós de uma rede LR-WPAN permanecem inativos durante longos períodos de tempo. No entanto, o protocolo IP assume que um dispositivo se encontra sempre conectado.

O uso de pacotes IPv6 em redes de baixo consumo baseadas em dispositivos IEEE802.15.4 requer a superação de um obstáculo básico: como trafegar um pacote IPv6, que requer uma MTU mínima de 1280 bytes, em um quadro IEEE 802.15.4, que tem tamanho limitado a 127 bytes. Para superar esse problema, o IETF definiu um formato de compressão de cabeçalhos de pacotes IPv6 para entrega em redes 6LoWPAN.

A compressão de dados do padrão 6LoWPAN foi originalmente especificada na RFC-4944; porém, o (*blip*), uma de suas implementações mais conhecidas, segue o formato de compressão descrito no documento "draft-ietf-6lowpan-hc-06", que trouxe algumas melhorias ao método de compressão original. Originalmente, o padrão só tratava de compressão stateless (não-sensível ao contexto) quando o endereço é designado ao nó automaticamente por ele mesmo, ou seja, ele é auto-configurado. Posteriormente, foi adicinada ao padrão a compressão (*statefull*) (sensível ao contexto) quando o endereço é delegado ao nó por outros meios de gerência da rede, por exemplo, usando o DHCPv6 nas redes IPV6.

Assim, um dos objetivos fundamentais do 6LoWPAN é definir a forma como um pacote IPv6, cuja (*MTU - Maximum Transmission Unit*) mínima é de 1280 bytes, deve ser acomodado em quadros IEEE 802.15.4, que têm MTU de apenas 127 bytes. Nesse sentido, o 6LoWPAN acrescenta uma nova camada, denominada de camada de adaptação ou camada 6LoWPAN, à pilha IP, que permite fragmentar e desfragmentar um pacote IPv6 em frames IEEE 802.15.4 e efetuar a compressão do cabeçalho IPv6. O padrão 6LoWPAN indica ainda os requisitos para efetuar o encaminhamento em redes (*mesh*), sugere algumas recomendações de segurança e descreve as alterações a efetuar a alguns protocolos da pilha IP de forma a otimizá-los para redes LR-WPAN. Uma alteração de particular importância é a encontrada no (*Neighbor Discovery Protocol*), um protocolo incluído na pilha TCP/IP que passa a aglutinar funções que eram executadas por protocolos distintos da pilha IPv4, como o ICPM e ARP: auto-configuração

de endereços; resolução de endereços; detecção de duplicação de endereços e descoberta de roteadores.

Algumas das principais características do LoWPAN são:

- Suporte a endereçamento MAC de 16 ou 64 bits
- Compressão de cabeçalhos eficiente
- Topologia baseada em estrela e malha.
- Cabeçalhos IPv6 base e de extensão, bem como cabeçalho UDP
- Auto-configuração da rede usando o protocolo ND
- Suporte a unicast, multicast e broadcast
- Suporte a roteamento IP (e.g. protocolo RPL)
- Suporte para uso de enlace mesh

Observe que como a RSSF é dinâmica, sendo caracterizada pela morte de alguns nós e entrada de outros a qualquer momento, a conectividade IP deve ser feita de maneira também dinâmica, através do suporte à atribuição dinâmica de endereços. Deve ser também tratado, o problema do grande número de endereços IP devido à existência de um número considerável de dispositivos na rede. Felizmente, ambas as características já possuem soluções provindas do IPv6.

Redes LoWPAN devem permitir topologias diversas, inclusive topologia em estrela e em malha. A topologia em malha deve permitir um roteamento *multi-hop* para os destinos, o que torna necessário que os dispositivos tenham a capacidade de "encaminhamento"dos pacotes. Neste sentido, alguns dos requisitos para os protocolos de roteamento em uma rede 6LoWPAN são:

1. Devido ao limite do tamanho do pacote, o protocolo de roteamento deve evitar desperdícios nos pacotes de dados, independente do número de saltos.

2. O protocolo de roteamento deve ter um baixo desperdício na troca de mensagens, independente da topologia empregada.

3. O protocolo de roteamento deve ter um baixo grau de computação e uso de memória, para que seja satisfeito o baixo custo e consumo de energia, característica da rede.

A topologia em estrela também deve possuir um conjunto de dispositivos responsáveis por fazer o encaminhamento das mensagens. Se for necessário ter outros tipos de interface de rede para obter isso, o desafio será integrar essas redes.

Cabe ressaltar que o tamanho máximo dos pacotes IPv6 é de 1280 octetos (bytes), o que faz com que um pacote IPv6 não caiba em um *frame* 802.15.4, que possui somente 127 octetos de tamanho máximo. Como o padrão 802.15.4 possui um desperdício máximo de 25 octetos, deve-se considerá-lo com apenas 102 octetos úteis. Soluções de segurança na camada de enlace desperdiçam 21 octetos no caso do AES-CCM-128, 13 octetos para o AES-CCM-64 e 9 octetos para o AES-CCM-32; sobrando somente 81 octetos disponíveis para a transmissão do pacote IPv6 e posteriores. Como o cabeçalho IPv6 possui 40 octetos de tamanho, restam apenas 41 octetos para os protocolos superiores, como o UDP, que utiliza um cabeçalho de 8 octetos, sobrando 33 octetos para os dados da aplicação. Com isso, é notável que deve-se empregar um mecanismo de fragmentação/remontagem dos pacotes, de forma que seja possível transmitir mais do que os 33 restantes em uma única mensagem.

Finalmente, observa-se que, embora a compressão seja um aspecto importante para o bom funcionamento da rede, a eficiência da rede pode depender de como será a camada de aplicação. Algumas abordagens de transmissão de XML como SOAP, por exemplo, se tornam inviáveis para redes LoWPAN, tornando necessária a criação ou modificações de protocolos de aplicação com o objetivo de adaptá-los a esse tipo de rede.

2.6 blip: The Berkeley Low-power IP stack for TinyOS

Entre as implementações existentes das recomendações do padrão 6LoWPAN para o sistema operacional TinyOS, uma das mais importantes é a blip (TINYOS-BLIP,). A implementação do blip disponibilizada atualmente está na versão 2.0, lançada em maio de 2010 para TinyOS 2.1.1, e apresenta conformidade total com a RFC 4944. A versão 2.0 usa compressão do cabeçalho 6lowpan/HC01 e inclui a implementação do IPv6 Neighbor Discovery Protocol, além de outras características adicionais, como seleção de rota default e roteamento ponto-a-ponto. O Blip foi implementado para as plataformas MicaZ, TelosB e Epic, e o seu código atual foi testado em redes de até 75 nós, para análise de estabilidade e desempenho.

O blip oferece suporte a diversas funcionalidades do IPv6. Dentre as mais importantes, pode-se destacar a descoberta de vizinhos e o roteamento ponto-a-ponto. Além disso, o protocolo tem suporte ao ICMP, UDP e TCP, embora a implementação do TCP ainda esteja em caráter experimental e possa não oferecer o desempenho e a confiabilidade usuais do TCP. A conexão entre a RSSF e a Internet é feita através de um driver de tunelamento para Linux. Desse modo, um computador executando o sistema operacional Linux opera como um roteador de borda, repassando os pacotes para a Internet. O driver também implementa o protocolo de descobrimento de vizinhos do IPv6, permitindo que os nós da rede encontrem o roteador de borda.

O endereçamento, a auto-configuração sem estados e a compressão de cabeçalho do blip é feito em conformidade com a RFC 4944. Como a rede de sensores utilizando blip pode ser vista como qualquer outra rede IP, muitos programas conhecidos para computadores também podem ser utilizados, dentre eles: ping6, tracrt6 e nc6.

Por se tratar de uma implementação para o sistema operacional TinyOS, o blip é desenvolvido utilizando a linguagem nesC. Assim, o Blip é facilmente separado em componentes. A Figura 2.8 mostra os componentes principais do código Blip 1.0, utilizado durante o desenvolvimento desse trabalho.

Os componentes da camada de rede são:

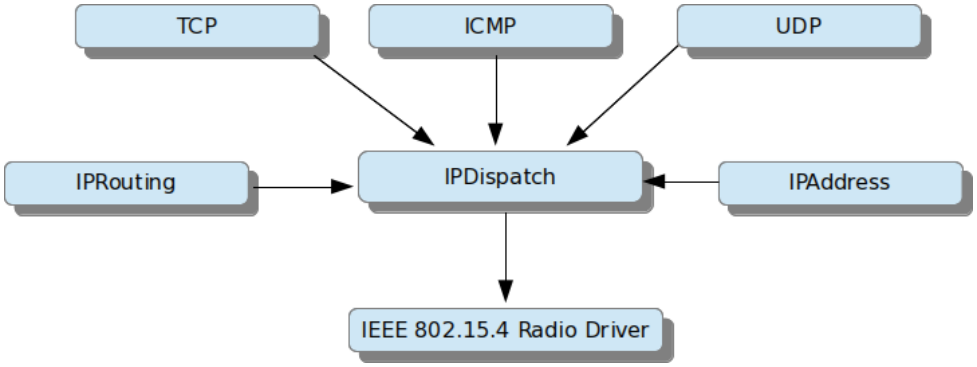


Figura 2.8: Componentes principais do Blip 1.0

- *IPRouting*: implementa a funcionalidade de descoberta de vizinhos do IPv6 (Neighbor Discover Protocol), constrói a tabela de roteamento e define o próximo nó intermediário na rota de um pacote;
- *IPDispatch*: trata as requisições de encaminhamento de pacotes recebidas da camada de transporte usando os serviços da camada de enlace, realiza a fragmentação de pacotes e a validação das transmissões;
- *IPAdress*: obtém os endereços IPv6 e IEEE802.15.4 no nó.

O componente ICMP (*Internet Control Message Protocol*) faz requisições de envio ao roteador. O componente UDP(*User Datagram Protocol*) é responsável pelos datagramas UDP e provê funcionalidades para facilitar o recebimento e envio de pacotes UDP. No componente TCP são implementadas funções de envio e recebimento de pacotes que visam garantir entrega confiável (esse componente funciona apenas de forma experimental e não implementa todas as definições do protocolo TCP).

Para funcionar, o *blip* requer que uma estação base conecte a RSSF a um computador. Tanto o computador quanto a estação base implementam a funcionalidade de roteador de borda. O código na estação base não precisa da camada IP e nem da camada de transporte, e acessa o driver do rádio diretamente. Assim, os pacotes são diretamente encaminhados para o computador e vice-versa.

Com relação ao roteamento, cada nó age como se fosse um roteador IP e deve manter uma rota em direção ao roteador de borda. Uma mensagem de solicitação do roteador é enviada

quando o dispositivo é iniciado e sempre que ocorre falha em uma das rotas. Adicionalmente, cada nó envia uma mensagem de aviso sempre que recebe uma mensagem de solicitação de rota ou quando a sua distância ao roteador de borda é modificada. O *blip* utiliza um sistema de acks para confirmar se o próximo nó recebeu a mensagem enviada.

2.7 Simuladores

Nesta seção são descritos os simuladores estudados durante a elaboração deste trabalho, juntamento com suas principais características.

2.7.1 The Network Simulator - ns2

O ambiente de simulação *Network Simulator 2*(NS-2..., 2012) é um simulador de eventos discretos, capaz de simular redes TCP (e variantes), protocolos de roteamento e de *multicast* para ambas redes cabeada e sem fio.

Apesar da grande utilização, o NS-2 possui uma curva de aprendizagem rígida até o usuário atingir uma plena familiaridade com o a linguagem de *scripting* e as técnicas de modelagem do simulador, que algumas vezes são difíceis e confusas de se escrever. Além do mais, como o NS-2 é um simulador de redes de uso geral, ele não possui algumas características fundamentais para RSSF. Por exemplo, o protocolo *Ethernet* é amplamente utilizado para redes LAN cabeadas, e o protocolo 802.11 é amplamente utilizado para redes LAN sem fio; porém, se tratando de RSSF, os protocolos são geralmente específicos em cada uma das aplicações e, portanto, não existe a grande maioria desses protocolos implementadas para o NS-2.

2.7.2 OMNet++

O *OMNet++*(OMNET++, 2012) é um *framework* e biblioteca C++ de simulação baseada em componentes, extensivo e modular. Esse simulador é voltado a uma gama maior de redes, que variam desde redes de comunica com e sem fio, redes *NoC*[6], redes em filas etc.

[6] *Network-on-chip*

A grande limitação do OMNet++ é que ele não representa a implementação real dos protocolos. Ao invés disto, ele fornece um framework genérico onde é possível implementar e avaliar o algoritmo para um primeiro teste e que, posteriormente, precisa ser escrito na linguagem da plataforma a ser utilizado.

2.7.3 TOSSIM

O simulador TOSSIM(LEVIS; LEE; WELSH, 2003) é o simulador nativo do TinyOS, e consegue simular uma rede de sensores TinyOS completa, com escalabilidade e alta fidelidade. Ele possui uma interface de programação (*framework*), onde é possível fornecer informações de sensoreamento aos nós da rede. A simulação no TOSSIM é somente pela interface de programação e sua principal desvantagem está no fato de ser possível simular nativamente somente uma rede utilizando um software por vez, o que torna inviável para simulação de 6LoWPAN, pois temos sempre no mínimo duas imagens, sendo uma para o nó aplicação (ex.: *UDPEcho*) e outra para a estação base. Além disto, o TOSSIM não permite a coleta do consumo de energia dos nos da rede, sendo necessário a utilização de sua extensão PowerTOSSIM.

2.7.4 Cooja

O simulador Cooja(ÖSTERLIND, 2006) é um simulador para o *Contiki*[7](CONTIKI...,) extensível pela utilização de interfaces e plugins. As Interfaces representam as propriedades de um nó sensor (ou outro dispositivo) como sua posição, o pressionar de um botão ou o transmissor de rádio. Já os plugins são utilizados para interagir com a simulação, podendo alterar sua velocidade, analisar o tráfego da rede etc.

2.7.5 Avrora

O Avrora(AVRORA..., 2012a; AVRORA..., 2012b) foi originalmente criado para ser uma suíte de ferramentas para análise e *debugging* de programas para microcontroladores AVR.

[7]O Contiki, assim como o TinyOS, é um *framework* de desenvolvimento para Redes de Sensores Sem Fio

Atualmente ele suporta a simulação de nós sensores Mica2 e Micaz, quer permite a simulação de redes de sensores com instrumentação dinâmica e análise estática.

Apesar de ter sido construído inicialmente para microcontroladores AVR, o Avrora tem algumas funcionalidades básicas para simulação do microcontrolador *MSP430* e, consequentemente, a simulação de redes com simuladores *TelosB*.

O *Avrora* foi escolhido para este trabalho por alguns motivos, como o fato de ele ser específico para redes de sensores, amplamente utilizado, *open-source*, com atualizações constantes[8], grupo de discussão com suporte de toda a comunidade que o utiliza e desenvolve (MAILING...,), fácil utilização, ampla documentação.

2.8 Segurança da Informação

A segurança da informação é a área da tecnologia da informação responsável por prover recursos e mecanismos que visam identificar, prevenir e solucionar problemas que ameaças podem fornecer aos ativos de uma organização. Quanto aos ativos, esses consistem em qualquer recurso que tenha valor para a organização, sejam equipamentos, ferramentas lógicas (como softwares), estruturas físicas, funcionários e até mesmo a informação em si. De maneira geral, a segurança da informação é responsável por proteger os bens valorados da empresa, além de fornecer mecanismos de contra medidas, caso algum dos recursos sofram algum tipo de ameaça ou ataque, visto que muitos sistemas de informação não foram projetados para serem seguros. Além disso a segurança da informação que pode ser alcançada por meios técnicos é limitada (International Organization for Standardization, 2005), o que torna necessário a criação de um sistema de gestão de segurança que altere os processos e rotinas da organização.

2.8.1 Plano de Segurança

A abordagem para a criação de um modelo de segurança pode ser dividida em 5 passos(FRASER, 1997):

[8] Última atualização em 02/01/2012

1. Identificar o que se está tentando proteger;
2. Determinar contra o que está tentando proteger;
3. Determinar qual a probabilidade da ameaça ocorrer;
4. Implementar mecanismos que protejam os ativos em um custo-benefício viável;
5. Revisar os processos continuamente e efetuar melhorias sempre que um ponto fraco for encontrado.

Os dois primeiros passos consistem em identificar quais ativos deseja-se proteger e quais são as possíveis ameaças que esses ativos estão sujeitos, definindo de acordo com sua disponibilidade, confidencialidade e integridade (FRASER, 1997). Um ativo é qualquer coisas que tenha valor para a organização (International Organization for Standardization, 2004a), seja computadores, dispositivos de rede, documentos e até funcionários.

2.8.2 Definições de Atributos

De acordo com (International Organization for Standardization, 2004a), os três principais atributos da segurança da informação podem ser definidos como:

Disponibilidade propriedade de estar acessível e utilizável sob demanda por uma entidade autorizada

Confidencialidade propriedade de que a informação não esteja disponível ou revelada a indivíduos, entidades ou processos não autorizados

Integridade propriedade de salvaguarda da exatidão e completeza de ativos.

Além dos três atributos citados, outras propriedades, tais como autenticidade, responsabilidade e não repúdio, podem também estar envolvidas (International Organization for Standardization, 2005). A autenticidade visa determinar se um determinado objeto é confiável ou não, no que diz respeito a sua origem. A responsabilidade vida definir as atribuições de um determinado indivíduo. O não repúdio objetiva garantir que os autores de um evento não neguem sua criação,

de forma que se for classificado como uma ameaça os responsáveis possam ser identificados de maneira objetiva.

Após a identificação e definição dos ativos da organização, é necessário identificar quais risco tais ativos estão sujeitos. Com isso, cada risco é examinado e então é definido qual o impacto que ele apresenta. Os riscos podem ser avaliados e, então, é possível classifica-los de acordo com sua importância. Essa classificação pode ser feita através da probabilidade do evento ocorrer e sua consequência (International Organization for Standardization, 2005).

Um ou uma série de eventos de segurança da informação indesejados ou inesperados, que tenham uma grande probabilidade de comprometer as operações do negócio e ameaçar a segurança da informação é definido como um incidente de segurança da informação (International Organization for Standardization, 2004b).

É fundamental que haja um equilíbrio entre os gastos com os controles de segurança e os danos causados ao negócio gerados pelas suas potenciais falhas de segurança (International Organization for Standardization, 2005), de forma que a organização não gaste mais recursos com a segurança do que gastaria em métodos corretivos de ataques.

2.9 Criptografia de Dados

O temo criptografia origina-se do Grego, onde, *kryptós* significa “escondido” e *gráphein* significa “escrita”. Em outra palavras, a criptografia é uma ciência que estuda os métodos e os princípios de se enviar uma mensagem, de forma que somente o destinatário consiga entendê-la, mesmo que a mensagem seja detida ou interceptada por indivíduos não autorizadas.

Existem dois métodos de criptografia que utilizam chaves, a criptografia simétrica, que utiliza somente uma única chave a criptografia assimétrica, que utiliza um par de chaves para cada um dos envolvidos na comunicação.

2.9.1 Criptografia Simétrica

Na criptografia simétrica, uma única chave é utilizada tanto para criptografar quanto para descriptografar. Apensar de os algoritmos assimétricos serem bastante rápido quanto ao processamento, é estritamente não recomendado sua utilização pura, pois ele necessita que a mesma chave seja compartilhada entre as pontas, o que a torna extremamente insegura caso o meio de comunicação não seja seja confiável.

Na Figura 2.11 é possível analisar o processo de envio de uma mensagem cifrada entre Alice e Bob utilizando um algoritmo simétrico de criptografia qualquer. Inicialmente Alice gera uma chave a ser utilização na cifragem/decifragem da mensagem original. Após ela enviar a mensagem cifrada, ela precisa de alguma forma compartilhar a chave gerada com Bob, para que ele consiga decifrar e ler a mensagem original. Em posse da mensagem cifrada e da chave gerada, Bob decifra a mensagem e consegue assim obter a mensagem original.

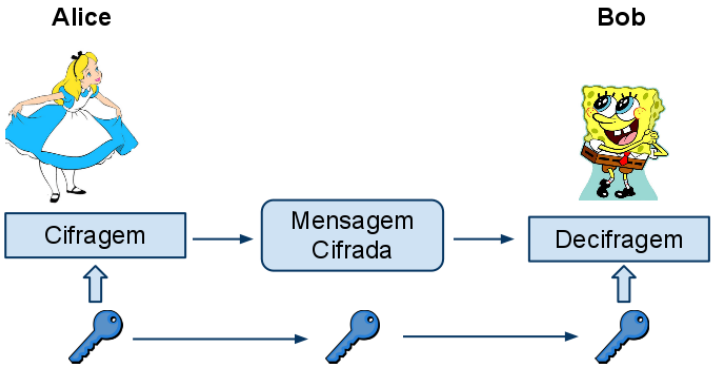


Figura 2.9: Criptografia simétrica em seu funcionamento normal

O grande problema dos algoritmos simétricos está no meio em que a chave é transmitida. Ao enviar a chave para Bob, um atacante pode interceptar a comunicação e, em posse da chave transmitida, conseguir decifrar a mensagem. Este cenário pode ser visto na Figura 2.10.

Apesar dos problemas causados pela insegurança da transmissão padrão da chave, os algoritmos de criptografia simétricos possuem desempenho melhor em relação aos algoritmos de criptografia assimétricos (vide seção 2.9.2).

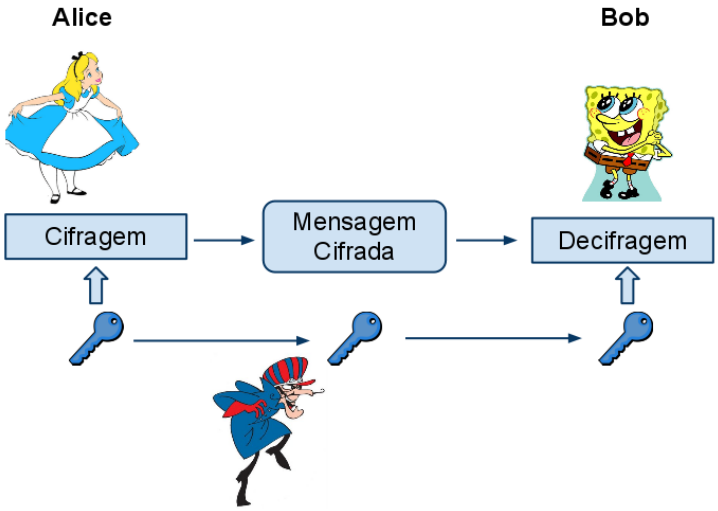


Figura 2.10: Interceptação da chave em algoritmos de criptografia simétrica

2.9.2 Criptografia Assimétrica

Na criptografia assimétrica existe um par de chaves para cada um dos que desejam comunicar. A principal característica dos algoritmos assimétricos está no fato de uma mensagem criptografada com uma chave pública só poder ser descriptografado utilizando a chave primária equivalente, assim como uma mensagem criptografada com uma chave privada só é descriptografada utilizando a chave pública equivalente.

Na Figura 2.11 é possível observar o processo utilizado para enviar uma mensagem de Alice para Bob. Inicialmente, Bob compartilha sua chave pública (azul). Alice, que obtém a chave pública de Bob, utiliza-a para cifrar mensagem que deseja transmitir. Dessa forma, ao chegar a mensagem cifrada para Bob, somente sua chave privada é capaz de decifrar e obter a mensagem original. Com isso, é possível manter um mecanismo de segurança, sem a necessidade de se preocupar em um atacante interceptar a chave pública transmitida por Bob.

2.10 Conclusões do Capítulo

Esta capítulo apresentou as principais tecnologias estudadas para o desenvolvimento da proposta apresentada nesta dissertação. O estudo do sistema operacional TinyOS e da sua linguagem de programação associada nesC, o entendimento da tecnologia de enlace IEEE 802.15.4, a

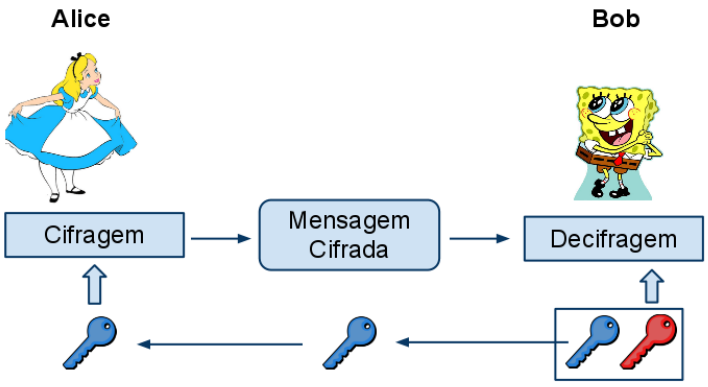


Figura 2.11: Criptografia assimétrica em seu funcionamento normal

investigação experimental da pilha 6LoWPAN e da sua implementação blip, bem como o estudo e a experimentação de ambientes de simulação para RSSF, constituíram a base teórica e prática para o desenvolvimento desta proposta de trabalho, explorada nos capítulos 4 e 5.

O tema de segurança constitui o ponto central de investigação desta dissertação e, nesse sentido, o próximo capítulo introduz alguns dos principais ataques em RSSF. Com base nos estudos de segurança em RSSF realizados ao longo desta dissertação, em especial o estudo da questão da autenticidade dos nós ingressantes na rede, é que foi proposto o do ambiente de experimentação apresentado nesta dissertação.

3 *Ataques em Redes de Sensores Sem Fio*

3.1 Introdução

Juntamente com o avanço das pesquisa para melhorar a capacidade e o tempo de vida das Redes de Sensores Sem Fio, diversas aplicações passam a trabalhar em ambientes hostis, tornando necessário a aplicação de mecanismos de segurança que, devido aos aspectos da rede, diferem dos métodos empregados nas redes de computadores tradicionais.

Além dos mecanismos de segurança, diversos mecanismos de ataque surgem no ambiente de Redes de Sensores Sem Fio. Tais métodos podem possuir objetivos diferentes, como a obtenção de dados sigilosos do monitoramento dos sensores ou a indisponibilização de parte ou toda rede. Entre esses métodos, os mais comuns são Ataques de Negação de Serviço (DoS[1]), ataque *Sybil* e *Wormhole*. Como modelos de prevenção e melhoria da segurança das Redes de Sensores Sem Fio, existem modelos de Sistemas de Detecção de Intrusão (IDS[2]).

Com isto, uma Rede de Sensores Sem Fio deve possuir alguns requisitos de segurança indispensáveis, como controle de acesso, para evitar tentativas não autorizadas de acesso a um nós; integridade da mensagem, para detectar e prevenir mudanças não autorizadas nas mensagens; confidencialidade, para garantir que o nó sensor criptografe as mensagens de forma que somente os nós que irão receber a mensagem consiga entende-las; e proteção contra replicação, para garantir que o nó sensor ofereça proteção caso um atacante reutilize uma mensagem autêntica para acesso à rede (POTDAR; SHARIF; CHANG, 2009).

[1] Do inglês, *Denial of Service*
[2] Do inglês *Intrusion Detection System*

A seguir, serão apresentado os principais ataques aplicados a Redes de Sensores Sem Fio e alguns métodos encontrados na literatura de detecção prevenção de tais ataques.

3.2 Comprometimento de Nós

Uma vez que um nó da Rede de Sensores é comprometida, a segurança de toda a rede se degrada rapidamente caso não existam contra medidas para lidar com esse tipo de evento (MATHEWS et al., 2007).

Ao gerar um evento de monitoramento, diversos nós sensores da rede enviam mensagens para a estação base. Essas mensagens muitas vezes são idênticas ou possuem diferenças irrelevantes. Para minimizar o *overhead* da rede devido ao grande tráfego gerado, as mensagens passam a serem combinadas em um simples pacotes, de forma a minimizar o fluxo da rede. Entretanto, sem um mecanismo de proteção contra nós comprometidos, a rede passa a ficar sujeita aos mais diversos ataques. Em (HU; CHEN, 2009), propõe-se um protocolo *Protecting Data Aggregation from Compromised Nodes (PDCAN)*, que utiliza funções *hash* para proteger a integridade dos pacotes, e um mecanismo de criptografia assimétrica para autenticar um novo nó entrante na rede.

Além de prevenir a inclusão de nós comprometidos na rede caso ocorra, é fundamental fornecer um mecanismo de identificação de tais nós. Em (MATHEWS et al., 2007), é apresentado um algoritmo de detecção de intrusão baseado em anomalias onde, com base nas características dos pacotes enviados pelo nó maléfico em comparação com o restante da rede, é possível verificar se este nó está enviando informações legítimas ou não. Quando uma vizinhança detectar alguma anomalia de um determinado nó, uma mensagem é enviada a estação base, que verifica sua consistência. Caso conclua que o nó está comprometido, uma mensagem de *broadcast* é enviada à rede informando a presença de um intruso. Ao receberem a mensagem, os nós limpam o *buffer* de quaisquer mensagens vindas do invasor e em seguida marcam o nó como "comprometido".

3.3 Negação de Serviço

O objetivo de um ataque de Negação de Serviço é deixar um recurso inacessível ou degradado para usuários legítimos do sistema(GILL; YANG, 2010). De acordo com (WALTERS et al., 2007), os ataques de Negação de Serviço são os mais notáveis em Redes de Sensores, pois além de poderem causar desde o congestionamento de um nó até a violação do protocolo *802.11 MAC*, quando bem efetuado, pode ser impossível de se proteger, pois um nó mais poderoso, pode facilmente bloquear o funcionamento normal de um nó sensor qualquer.

Os ataques de Negação de Serviço podem ser efetuados em qualquer uma das camadas de comunicação (física, elnaçe, rede, transporte e aplicação). Cada uma das camadas possuem suas formas de ataque e seus mecanismos de defesa(RAYMOND; MIDKIFF, 2008).

Na camada física, a rede é sujeita a ataques de congestionamento e a ataques de adulteração ou destruição de nós sensores. Para o primeiro, as principais técnicas de defesa são "detectar e dormir" e fazer roteamento em volta da região congestionada. No segundo tipo ataque, sugerem métodos de camuflagem de nós sensores e embalagem a prova de violação física.

Na camada de acesso ao meio, tem-se os ataques de "interrogação" e os ataques de "negação de dormir". O ataque de interrogação explora o processo "requisição para enviar"[3] e "livre para enviar"[4] presentes em muitos protocolos de acesso ao meio. Um atacante pode indisponibilizar os recursos de um nó enviando repetidas mensagens de requisição, de forma que extraia resposta "livre para enviar" de um nó vizinho(RAYMOND; MIDKIFF, 2008). Uma das formas de proteger-se desse ataque é autenticação e proteção contra reenvio de mensagens. Outra forma de ataque na camada de acesso ao meio é o chamado "negação de sono", que é executada ao congestionar a rede com um determinado tipo de pacote que impeça outros nós de dormirem. Para proteger-se desse ataque, deve-se implementar autenticação e proteção contra reenvio de mensagem e, para minimizar o impacto pode-se detectar e dormir, além de implementar algum mecanismo de proteção contra ataques de *broadcast*.

[3] RTS do inglês – *request-to-send*
[4] CTS do inglês – *clean-to-send*

Na camada de rede, diversos ataques podem ser efetuados com o objetivo de indisponibilizar a rede. Dentre esses ataques podemos citar a replicação de nós, a alteração de rota e a alteração do controle de tráfego. Esses ataques podem ser evitados utilizando autenticação, formação de *clusters*, roteamento geográfico, entre outras técnicas.

Na camada de transporte tem-se os ataques de inundação de mensagens de sincronização[5], que podem ser defendidas através da utilização de *SYN Cookies*. Esse consiste em codificar informações dos clientes nas mensagens e enviar de volta. De acordo com (RAYMOND; MIDKIFF, 2008), essa técnica é considerada indesejável para Redes de Sensores Sem Fio. Além disto, também existe o ataque de dessincronização, que é protegido através da autenticação de pacotes.

Em (GILL; YANG, 2010), é apresentado o design e a implementação de uma outra estratégia de defesa, juntamente com as medidas de proteção existentes, de forma que supere as insuficiências geradas pelas metodologias atuais. É apresentado o ataque de Negação de Serviço (DoS) e Negação de Serviço Distribuído (DDoS) em duas categorias gerais:

1. Os ataques criam uma quantidade relativamente pequena de de conexões com o alvo, e
2. os ataques simplesmente tentam indisponibilizar o serviço inundando o alvo com múltiplas conexões simultâneas

Em (GILL; YANG, 2010), para proteção de ataques de Negação de Serviço Distribuído, é definido um modelo de proteção dividido em dois mecanismos principais:

o método de detecção que consiste em identificar se está ocorrendo um ataque no alvo,
e o método de resolução que vista extinguir e proteger a rede de um possível atacante.

3.4 Sybil

Uma Rede de Sensores sem Fio é vulnerável aos mais diversos tipos de ataques. Por exemplo, um atacante pode inserir um nó maléfico na rede, ou comprometer um determinado nó,

[5] *SYN Flood*

de forma indisponibilizá-lo. Para prevenir-se desse tipo de ameaça, são criados mecanismos de redundância, onde um conjunto de nós são alocados para a mesma tarefa, de forma que a indisponibilidade de algum determinado nó seja suprida pelo restante da rede.

O ataque *sybil* ocorre quando um nó sensor maléfico falsifica a identidade de um número ilimitado de nós da rede para ferir o mecanismo de redundância de rede. Em (WALTERS et al., 2007) é citado uma definição do nó *sybil* como um "dispositivo malicioso que, ilegitimamente, assume múltiplas identidades".

Em (NEWSOME et al., 2004), é feita uma taxinomia do ataque *Sybil*, classificando-o em três dimensões. A dimensão I diz respeito ao tipo de comunicação, podendo ser direta ou indireta. Na comunicação direta, o nó maléfico comunica-se diretamente com os nós legítimos, enquanto na comunicação indireta a mensagem enviada pelo nós malicioso é roteada até o nó destino.

Na dimensão II, o ataque é classificado quanto ao tipo de identificação, podendo ser gerada ou roubada. Na identificação gerada, os nós *sybil* assumem identidades arbitrárias, de forma que a rede passe a acreditar que possui uma topologia densa quando, na verdade, a maioria das identidades são geradas pelo nó maléfico. O ataque por identidade roubada ocorre quando um nó maléfico identifica um ou mais nós da rede e então replica seus códigos de identificação.

Na dimensão III, o ataque é classificado como simultâneo ou não simultâneo. No ataque simultâneo o nó maléfico tenta copiar todas as identidades dos sensores da rede, enquanto que no não simultâneo o atacante obter um conjunto de identidades por um período de tempo.

De acordo com (ZHANG et al., 2005), os mecanismos de redundância são baseados em identificação, onde cada nó possui um identificador único. Dessa forma, a rede está vulnerável a ataques em que se falsificam a identidade de nós, como o ataque *sybil*. Ainda em (ZHANG et al., 2005), é proposto um método de proteção contra ataques *sybil* baseado na distribuição de chaves secretas, onde cada identidade a associada com sua chave única. A ideia básica é assinar cada nó com uma identidade única e criar um certificado de validação dessa identidade.

Em (MURALEEDHARAN; YAN; OSADCIW, 2007), é apresentado um mecanismo de

detecção de ataques *sybil* baseado em inteligência cognitiva. Nele é apresentado um protocolo cognitivo que se baseia na inteligência de insetos que atuam de maneira coletiva.

3.5 Wormhole

Em ataques de *wormhole*, um nó sensor envia um pacote ou um conjunto de pacotes de forma a deturpar o protocolo de roteamento empregado na rede, tornando ineficiente o processo de descoberta de vizinhança na rede (WALTERS et al., 2007).

Em (ZHAO et al., 2010), é apresentado um método de detecção de *wormhole* através de análise estática. O algoritmo proposto consegue detectar e localizar os *wormholes* na rede com sucesso e eficiência. O funcionamento do algoritmo consiste em fazer um roteamento por vários caminhos e, através de uma análise estática e algumas restrições de tempo, identificar as ligações suspeitas.

3.6 Exploração de Vulnerabilidades

Além dos ataques visto, um nó sensor está sujeito a falhas de implementações que permitem explorar o nó e executar código arbitrário. O grande desafio está no fato de os nós sensores rodarem com microcontroladores e microprocessadores que possuem arquitetura harvard, o que dificulta a exploração de falhas como *buffers overflows* desafiadoras.

Nos meios de exploração comuns de computadores, um facilitador é a arquitetura *Von Neumann* onde o código de programa e os dados compartilham a mesma memória. Portanto, nativamente, é possível inserir código de máquina como dados e induzir (através da exploração de alguma falha) o nó sensor a executa-lo. Porém, na arquitetura *Harvard*, uma das grandes diferenças está no fato de o código do programa e os dados estarem em memória separadas, portanto, por definição, é impossível executar um dado como se fosse programa.

Em (ASGAONKAR, 2010) e (FRANCILLON; CASTELLUCCIA, 2008), é descrito um método de ataque de injeção nós sensores que possuem arquitetura harvard. Para isto, utilizou-

se a técnica de programação orientada a retorno (ROP[6], que consiste em gerar uma falsa pilha de retornos, de modo a induzir o nó sensor a executar porções de códigos de sua própria implementação.

3.7 Conclusão

Este capítulo visou mostrar que assim como as redes de computadores convencionais, as Redes de Sensores Sem Fio estão sujeitas aos mais diversos ataques que vão desde o ataque físico com o comprometimento do nó; ataques de negação de serviço, que são extremamente de serem evitados devido a fragilidade e ao baixo recurso de processamento disponível nos nós; ataques de falsificação de endereços em um nó que visa desde o distúrbio do protocolo de roteamento (ataque *Wormhole*) ou visa influenciar nos mecanismos de redundância da rede (ataque *Sybil*).

[6]do Inglês *Return-oriented programming*

4 Ambientes de Simulação e Experimentação Simulado

4.1 Introdução

Neste capítulo é apresentado o ambiente de simulação e ferramentas auxiliares utilizadas para a obtenção de um ambiente de experimentação, onde é possível criar uma Rede de Sensores Sem Fio com um dada topologia, analisar seu desempenho no que diz respeito ao tempo de associação e ao consumo de energia. Na Seção 4.2 é apresentado a arquitetura do ambiente de simulação proposto, descrevendo a funcionalidade e a utilização de cada um de seus componentes. Na Seção 4.3 é apresentado os passos para a criação de um nós *Sybil*, sua utilização e o resultado de sua execução em uma rede, sendo avaliado no ambiente de simulado proposto. A Seção 4.4 conclui o capítulo.

4.2 Ambiente de Simulação

O ambiente de simulação propostao representa uma arquitetura de comunicação entre diversas aplicações. Sua arquitetura pode ser vista na Figura 4.1.

O ambiente de simulação possui duas partes bem definidas. Uma delas consiste no bloco que representa o simulador *Avrora*, e outro bloco que representa a estação base, e fornece acesso aos componentes internos às aplicações dos nós sensores abstraindo-se de estarem sendo executados em um ambiente simulado.

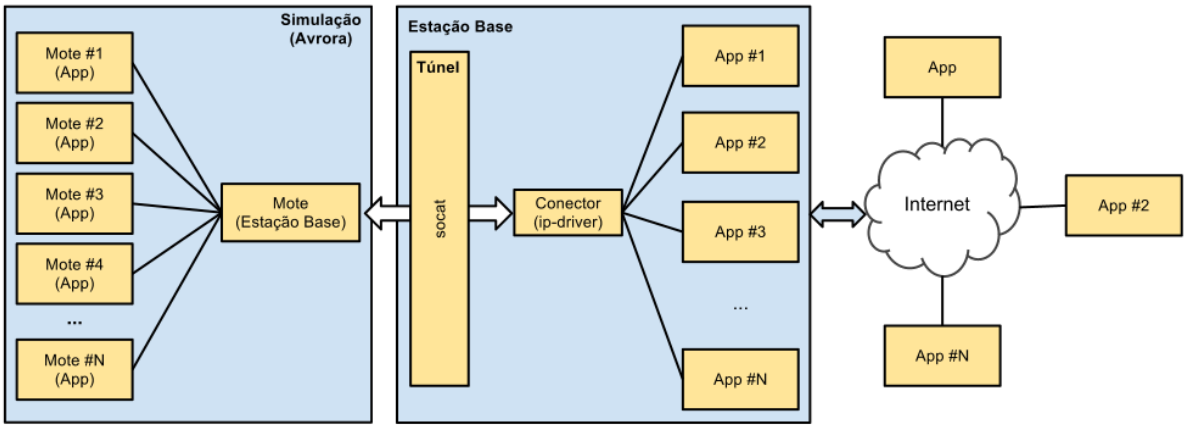


Figura 4.1: Arquitetura do Ambiente de Simulação

4.2.1 Bloco "Simulação"

O bloco da simulação é responsável por simular os dispositivos sensores da aplicação de monitoramento, e um dispositivo que fornecerá um meio de comunicação serial com a estação base. Como teste para aplicações *TinyOS* rodando sob a pilha IP Blip 1.0 podemos ter a aplicação *UDPEcho* ou *TCPEcho* como programas executados nos motes, e a aplicação *IPBaseStation* como programa do mote que comunicará com a estação base. Tudo o que está dentro do bloco de simulação é executado dentro de um ambiente virtual.

Abaixo segue um exemplo da execução do *Avrora* para a simulação de uma rede com 10 nós, sendo 9 deles para aplicação nos nós (*UDPEcho*) e 1 para estação base (*IPBaseStation*).

```
$ avrora -actions=simulate -simulation=sensor-network \
> -monitors=leds,packet,serial,energy,real-time \
> -topology=static -topology-file=topologia.top \
> -nodecount=1,9 IPBaseStation.elf UDPEcho.elf
```

Esse comando informa que deve ser simulado uma rede de sensores (primeira linha); que deve haver monitoramento dos leds, consumo de energia, dados que trafegam pelo rádio e serial, e que a execução deve ser feita em tempo real (segunda linha); que a rede terá uma topologia estática e de acordo com o arquivo *topologia.top* (terceira linha); e que serão dez nós, sendo um para o programa *IPBaseStation.elf* e nove para o programa *UDPEcho.elf* (quarta linha).

4.2.2 Bloco "Estação Base"

O bloco que representa a estação base consiste em uma máquina PC que executa diversas aplicações, como a aplicação que conecta ao nó sensores de borda do ambiente simulado, até o software conector que cria uma interface IP de comunicação com a rede interna.

Tunelamento (socat)

Como ferramenta de tunelamento foi utilizado o *socat*, um utilitário que estabelece uma comunicação bi-direcional, fornecendo diversos mecanismos de comunicação como TCP, IPv4, IPv6, pipe, pty etc. Com essa ferramenta é possível conectar-se à interface TCP/IP fornecida pelo *Avrora* e então criar um dispositivo virtual (*Virtual Device*) serializado, para que o acesso seja semelhante ao acesso direto a um dispositivo USB.

O comando *socat* a seguir é um exemplo de execução:

```
$ socat pty,link=/tmp/vty0,waitslave tcp:localhost:2390
```

Onde o primeiro parâmetro informa que deve ser criado uma interface serial (pseudo-terminal) no arquivo */tmp/vty0*, e esperar a conexão nesse dispositivo para conectar no segundo parâmetro, que representa uma conexão *TCP* em *localhost* e na porta *2390* de interface com o *Avrora*

Conector (ip-driver)

O conector utilizado foi o *ip-driver* fornecido com TinyOS (Blip 1.0). Ele é responsável por conectar em uma interface serial do sensor (no caso do ambiente de experimentação, a interface fornecida pelo *socat*) e criar uma dispositivo de comunicação IPv6, com o IP definido no arquivo de configuração.

Um exemplo de arquivo de configuração pode ser visto abaixo:

```
log INFO
addr fec0::100
proxy lo
channel 15
```

O parâmetro *log* informa o nível de informações que será exibida na saída, podendo ser *DEBUG*, *INFO*, *WARN*, *ERROR* e *FATAL*. O parâmetro *addr* indica qual deve ser o endereço da interface IPv6 local, que será utilizada para acesso à rede de sensores do ambiente simulado. O parâmetro *proxy* informa qual o dispositivo será utilizado para descoberta de vizinhança, podendo ser um dispositivo de rede (*eth0*, por exemplo), permitindo que outras máquinas utilizem a máquina atual como rota.

Após a execução do *ip-driver*, é possível visualizar as associações dos nós sensores à rede.

```
# driver/ip-driver /tmp/vty0 micaz
2012-05-30T17:34:45.602BRT: INFO: Read config from './serial_tun.conf'
2012-05-30T17:34:45.616BRT: INFO: Using channel 15
2012-05-30T17:34:45.616BRT: INFO: Retries: 5
2012-05-30T17:34:45.616BRT: INFO: telnet console server running on port 6106
2012-05-30T17:34:45.617BRT: INFO: created tun device: tun0
2012-05-30T17:34:50.750BRT: INFO: interface device successfully initialized
2012-05-30T17:34:50.750BRT: INFO: starting radvd on device tun0
2012-05-30T17:34:51.246BRT: INFO: Starting to proxy for 0x3
2012-05-30T17:34:51.335BRT: INFO: Starting to proxy for 0x2
2012-05-30T17:34:51.441BRT: INFO: Starting to proxy for 0x1
2012-05-30T17:34:51.453BRT: INFO: Starting to proxy for 0x5
2012-05-30T17:34:51.466BRT: INFO: Starting to proxy for 0x6
2012-05-30T17:34:51.550BRT: INFO: Starting to proxy for 0x4
2012-05-30T17:34:51.561BRT: INFO: Starting to proxy for 0x8
2012-05-30T17:34:54.755BRT: INFO: Starting to proxy for 0x7
```

O simulador, por padrão, define o identificador do nó como sufixo do endereço *IPv6*, ou seja, um nó com identificador *0x8* terá o *IPv6 fec0::8* associado à ele. Com isso, é possível trocar mensagens entre a estação base e os nós. Abaixo segue uma demonstração do *ping6* (pelo protocolo *ICMPv6*) da estação base (endereço *fec0::100*) para o nós com endereço *fec0::8*.

```
[maycon@darkside ~]$ ping6 fec0::8
PING fec0::8(fec0::8) 56 data bytes
64 bytes from fec0::8: icmp_seq=1 ttl=65 time=568 ms
64 bytes from fec0::8: icmp_seq=2 ttl=65 time=573 ms
64 bytes from fec0::8: icmp_seq=3 ttl=65 time=650 ms
64 bytes from fec0::8: icmp_seq=3 ttl=65 time=777 ms (DUP!)
64 bytes from fec0::8: icmp_seq=4 ttl=65 time=506 ms
```

```
64 bytes from fec0::8: icmp_seq=5 ttl=65 time=555 ms
64 bytes from fec0::8: icmp_seq=6 ttl=65 time=589 ms
64 bytes from fec0::8: icmp_seq=7 ttl=65 time=524 ms
64 bytes from fec0::8: icmp_seq=8 ttl=65 time=660 ms
64 bytes from fec0::8: icmp_seq=8 ttl=65 time=777 ms (DUP!)
64 bytes from fec0::8: icmp_seq=9 ttl=65 time=585 ms
^C
--- fec0::8 ping statistics ---
10 packets transmitted, 9 received, +2 duplicates, 10% packet loss, time 8998ms
rtt min/avg/max/mdev = 506.973/615.484/777.622/87.875 ms
[maycon@darkside ~]$
```

Aplicações Locais e Remotas

Como os nós são acessíveis através de endereçamentos IPv6, é possível criar aplicações que comuniquem com esses nós. Tais aplicações podem trabalhar com troca de mensagens UDP ou TCP (por se tratar do *Blip 1.0*), independente da linguagem utilizada para desenvolve-las.

Além do mais, como a estação base representa uma máquina e as aplicações são acessíveis através de mensagens IPv6, é possível que aplicações remotas acessem o ambiente simulado, tornando viável a utilização de máquinas com mais recursos para criar e gerencia o ambiente simulado e máquinas ou dispositivos móveis menos potente para as aplicações.

Uma grande vantagem dessa proposta está na criação de *testbed* simulados, onde remotamente teríamos máquinas com capacidade de processamento alto para executar o ambiente simulado, e uma outra máquina com menor processamento poderia simplesmente comandar e/ou coletar dados da rede monitorada.

4.3 Simulando o Ataque Sybil

Como visto no Capítulo 3 diversos ataques a RSSF são feitos através da falsificação de identidade dos nós sensores. Como prova de conceito foram feitas algumas modificações tanto na aplicação *UDPEcho* como em alguns componentes do TinyOS/Blip a fim de simular o ataque *Sybil* no ambiente de experimentação.

4.3.1 Modificações no Código

O código foi modificado de maneira onde a diretiva *SYBIL_ATTACK* ative as rotinas e características do ataque, não influenciando nas aplicações que não desejam fazer uso de tais recursos.

Para implementar o ataque foi utilizado uma instância do componente *TimerMilliC* e uma instância do componente *IPAddressC*. A primeira tem a função definir o intervalo em que o endereço do nó será alterado para que ele possa criar multiplas associações na rede, através da alteração do identificador do nó. Já o componente *IPAddressC* é responsável por alterar o endereço propriamente dito. Esse trecho de código pode ser visto na Figura 4.2 e na Figura 4.3.

```
58 #ifdef SYBIL_ATTACK
59   // Sybil attack components
60   components IPAddressC;
61   UDPEchoP.IPAddress -> IPAddressC;
62   UDPEchoP.SybilTimer -> TimerMilliC;
63 #endif
```

Figura 4.2: Alteração no *apps/UDPEcho/UDPEchoP.nc* para ataque Sybil (Parte 01)

```
38 module UDPEchoP {
39     uses {
40         interface Boot;
41         interface SplitControl as RadioControl;
42 
43         interface UDP as Echo;
44         interface UDP as Status;
45 
46         interface Leds;
47 
48         interface Timer<TMilli> as StatusTimer;
49 
50         interface Statistics<ip_statistics_t> as IPStats;
51         interface Statistics<udp_statistics_t> as UDPStats;
52         interface Statistics<route_statistics_t> as RouteStats;
53         interface Statistics<icmp_statistics_t> as ICMPStats;
54 
55         interface Random;
56 
57 #ifdef SYBIL_ATTACK
58         interface Timer<TMilli> as SybilTimer;
59         interface IPAddress;
60 #endif
61 
62     }
63 
64 } implementation {
```

Figura 4.3: Alteração no *apps/UDPEcho/UDPEchoP.nc* para ataque Sybil (Parte 02)

Durante a inicialização do nó sensor pelo evento *Boot.booted* o temporizador responsável por definir o intervalo de mudança do identificador do nó é inicializado, como visto na Figura4.4. Lembrando que todo código responsável pela funcionalidade do ataque *Sybil* está contido dentro da diretiva de compilação *SYBIL_ATTACK*.

```
70    event void Boot.booted() {
71        call RadioControl.start();
72        timerStarted = FALSE;
73
74        call IPStats.clear();
75        call RouteStats.clear();
76        call ICMPStats.clear();
77        printfUART_init();
78
79 #ifdef REPORT_DEST
80        route_dest.sin6_port = hton16(7000);
81        inet_pton6(REPORT_DEST, &route_dest.sin6_addr);
82        call StatusTimer.startOneShot(call Random.rand16() % (1024 * REPORT_PERIOD));
83 #endif
84
85        dbg("Boot", "booted: %i\n", TOS_NODE_ID);
86        call Echo.bind(7);
87        call Status.bind(7001);
88
89 #ifdef SYBIL_ATTACK
90        call SybilTimer.startPeriodic(SYBIL_INTERVAL);
91 #endif
92    }
```

Figura 4.4: Alteração no *apps/UDPEcho/UDPEchoP.nc* para ataque Sybil (Parte 03)

Quando o temporizador *SybilTimer* for acionado através do evento *SybilTimer.fired*, o nó chama o comando *setShortAddr* do componente *IPAddress*, passando como parâmetro um valor randômico maior ou igual a 0x2000 (ou 8192 em decimal), e menor que 0x3000 (ou 12288 em decimal), que será definido como novo sufixo do endereço IPv6 associado ao nó.

```
130 #ifdef SYBIL_ATTACK
131     event void SybilTimer.fired() {
132         call IPAddress.setShortAddr(0x2000 + (call Random.rand16() % 0x1000));
133     }
134 #endif
```

Figura 4.5: Alteração no *apps/UDPEcho/UDPEchoP.nc* para ataque Sybil (Parte 04)

Com isso a aplicação efetuará diversas associações na rede com identificadores entre 0x2000 e 0x3000. Essa definição foi feito somente para fins de testes, podendo se abranger a um conjunto/vetor fixo de identificadores. A associação do nó à rede será feita automaticamente pela implementação do *Blip*.

Como o nó altera sua identificação diversas vezes, seria um problema outros nós tentarem associar-se à ele com o intuito de utilizá-lo como nó de rota. Isso ocorre pois a implementação do *Blip* verifica se o identificador de destino de salto[1] de uma mensagem pertence ao nó atual ou não. Como o nó só possui uma única identificação por vez, foi necessário alterar a biblioteca do *Blip* a fim de que ele aceite qualquer mensagem que seja direcionada ou encaminhada a alguns dos endereços falsos do nó. Para isso foi necessário a modificação vista na Figura 4.6.

[1] variável *target_hop* na Figura 4.6

```
412  void updateSourceRoute(ieee154_saddr_t prev_hop, struct ip6_route *sh) {
413    uint16_t my_address = call IPAddress.getShortAddr();
414    uint16_t target_hop = sh->hops[ROUTE_NENTRIES(sh) - sh->segs_remain];
415    if ((sh->type & ~IP6ROUTE_FLAG_MASK) == IP6ROUTE_TYPE_INVAL || sh->segs_remain == 0) return;
416
417 #ifdef SYBIL_ATTACK
418    if (target_hop < 0x2000) {
419 #else
420    if (target_hop != htons(my_address)) {
421 #endif
422      printfUART("invalidating source route\n");
423
424      if (ROUTE_NENTRIES(sh) >= 2) {
425        sh->hops[0] = htons(prev_hop);
426        sh->hops[1] = target_hop;
427      }
428      sh->type = (sh->type & IP6ROUTE_FLAG_MASK) | IP6ROUTE_TYPE_INVAL;
429    } else {
430      sh->hops[ROUTE_NENTRIES(sh) - sh->segs_remain] = htons(prev_hop);
431      sh->segs_remain--;
432      printfUART("updating source route with prev: 0x%x remaining: %i\n",
433                 prev_hop, sh->segs_remain);
434    }
435  }
```

Figura 4.6: Alteração no *tos/lib/net/blip/IPDipatchP.nc* para ataque Sybil

4.3.2 Resultados

Como avaliação dos resultados foi feito a simulação de ambos os ambientes e, sem seguida, analisado como foi gerado a tabela de roteamento em cada um dos casos.

Primeiramente foi simulado uma rede com 10 sensores distribuídos linearmente, sendo um nós de comunicação com a estação base (*IPBaseStation*) e nove nós com a aplicação (*UDPEcho*). O seguinte comando *Avrora* foi utilizado para simular a rede:

```
[maycon@darkside Sim]$  avrora -actions=simulate -simulation=sensor-network \
> -monitors=serial,real-time -topology=static -topology-file=topologia.top \
> -nodecount=1,9 IPBaseStation.elf UDPEcho.elf
```

Tendo o arquivo de topologia *topologia.top* o seguinte conteúdo, onde cada linha representa um nó, e cada nó possui suas coordenadas espaciais X, Y e Z:

```
node0 0 0 0
node1 10 0 0
node2 20 0 0
node3 30 0 0
node4 40 0 0
node5 50 0 0
node6 60 0 0
node7 70 0 0
node8 80 0 0
node9 90 0 0
```

Após a execução das demais aplicações de simulação (*socat* e *ip-driver*), o resultado ao término da associação de todos os nós é o descrito na Figura 4.7. O comando **routes** do *Console Server* exibe o caminho que uma mensagem segue até atingir a estação base. A árvore de rotas pode ser vista de maneira mais amigável na Figura 4.8.

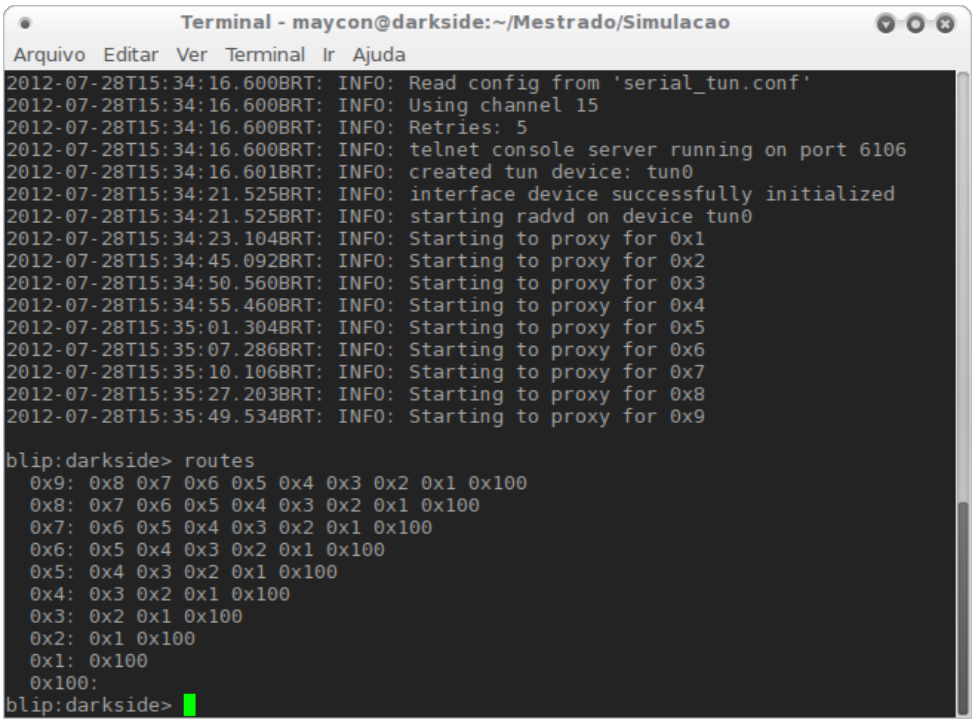


Figura 4.7: Tela do *Console Server* para uma rede normal

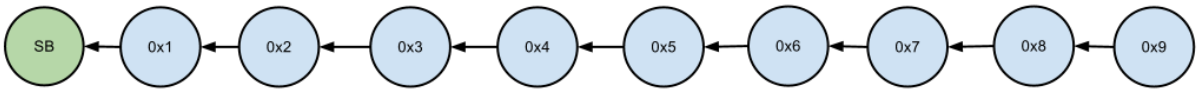


Figura 4.8: Roteamento da rede normal

Após a execução de uma aplicação, foi feito o mesmo teste substituindo o nó com identificação 5 (cinco) pelo nó que efetuará o ataque *Sybil*. Portanto existirão ainda dez nós na rede, sendo 1 para a estação base, seguido de quatro nós para uma aplicação normal (*UDPEcho*) seguido do nó *Sybil* com identificador maior ou igual a 0x2000 (ou ga) e de mais quatro nós de aplicação normal. A execução do *Avrora* para esta simulação pode ser visto abaixo:

```
[maycon@darkside Sim]$  avrora -actions=simulate -simulation=sensor-network \
> -monitors=serial,real-time -topology=static -topology-file=topologia.top \
> -nodecount=1,4,1,4 IPBaseStation.elf UDPEcho.elf UDPEcho-Sybil.elf UDPEcho.elf
```

Com isso, a associação da rede se faz naturalmente e é possível notar na Figura 4.9 o nó *Sybil* associando-se à rede com identificador *0x2392*. Com a implementação do nó *sybil* em execução visa alterar o identificação do nó e tentar uma reassociação, obtém-se o resultado esperado ao visualizar os nós *0x2c48* e *0x2edc* associarem-se à rede (Figura 4.9). Todas as três identificações representam o mesmo nó que efetuou o ataque *Sybil*.

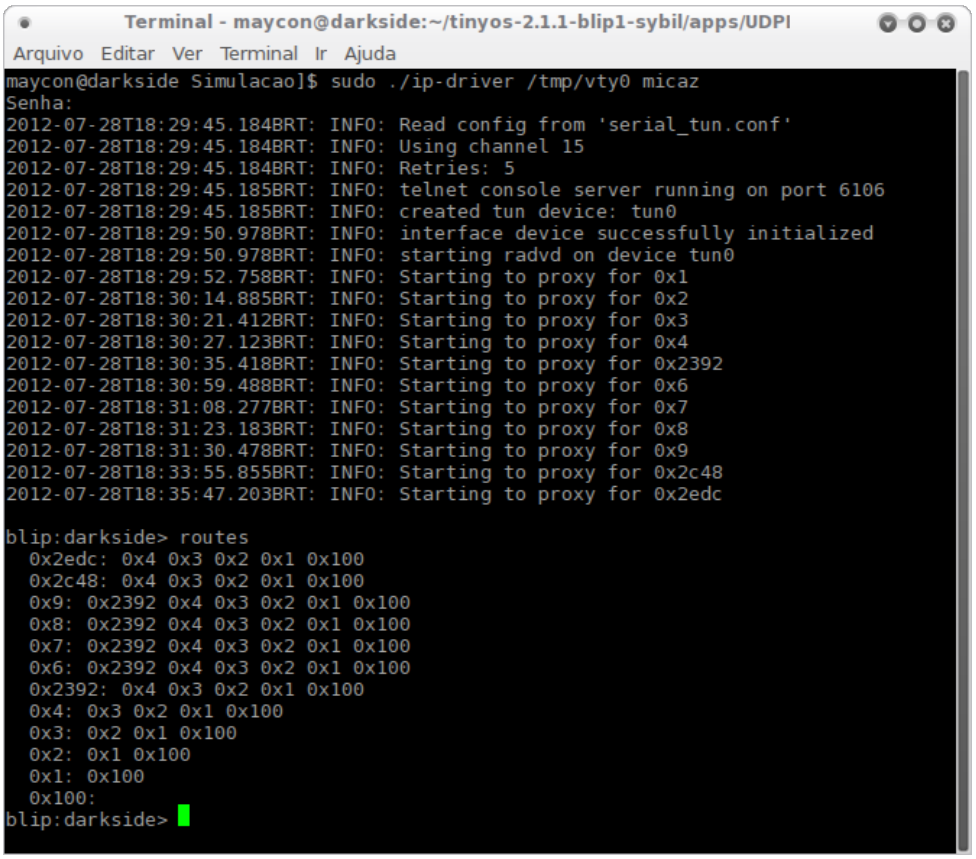


Figura 4.9: Tela do *Console Server* para uma rede sob ataque Sybil

O comando **routes** do *Console Server* na Figura 4.9 exibe os nós que passam através do nó *Sybil*. Essa lista de rotas representa a árvore de roteamento de cada um dos nós da rede, e pode ser vista na Figura 4.10.

4.4 Conclusões do Capítulo

Este capítulo apresentou um conjunto de simuladores com suas respectivas características e justificou o porque da utilização do Avrora para um ambiente de experimentação que execute a implementação do 6lowpan para Redes de Sensores Sem Fio no *framework* TinyOS. Em

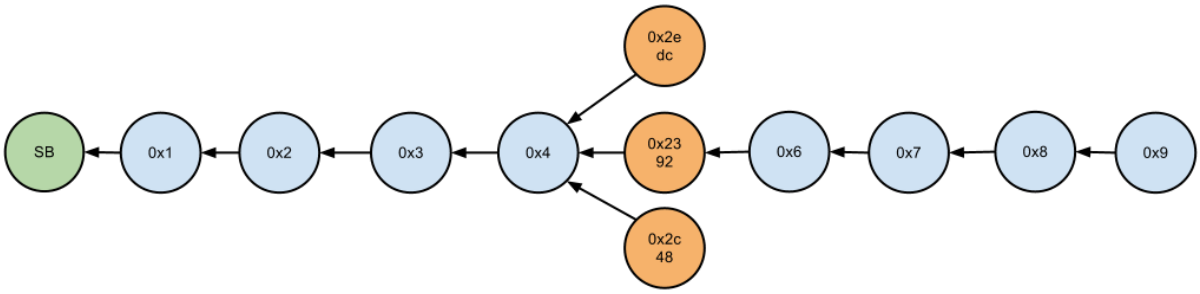


Figura 4.10: Roteamento da rede sob ataque Sybil

seguida foi apresentado um ambiente de experimentação simulada e seus componentes. Como teste, foi simulado uma rede em funcionamento normal, e uma rede que contém um nó maléfico efetuando o ataque *Sybil*.

5 *Módulo de Autenticação*

5.1 Introdução

Uma das proposta desse trabalho consiste em um módulo de autenticação para Redes de Sensores Sem Fio. Esse módulo visa fornecer um mecanismo onde, através de uma chave pré-compartilhada, um nó possa autenticar-se a um nó que possua a mesma chave. Essa chave pré-compartilhada deve ser armazenada de maneira secreta e, mesmo após o comprometimento físico de um nós, alguns mecanismos são propostos a fim de evitar o vazamento da mesma.

Na Seção 5.2 é apresentado o processo de autenticação proposto, descrevendo o funcionamento normal em cada um dos passos. Na Seção 5.3 é apresentado um mecanismo de proteção para a chave pré-compartilhada (PSK) baseado no armazenamento em memória volátil. Na Seção 5.4 é feito uma validação informal da proposta, onde a mesma é submetida em várias situações atípicas a fim de obter a chave pré-compartilhada ou autenticar-se a rede sem conhecê a chave pré-compartilhada. Na Seção 5.5 é apresenta os parâmetros de configuração do componente que permitem a configuração das funcionalidades do módulo. Na Seção 5.7 é apresentado a estrutura de componentes criada, seus objetivos e suas ligações. Na Seção 5.6 é apresentado o formato dos cabeçalhos criado tanto para comunicação quanto para armazenamento da chave pré-compartilhada, descrevendo cada um de seus campos. Na Seção 5.8 é feito uma avaliação experimental de uma aplicação com e sem a utilização do módulo de autenticação proposto, apresentado como resultado suas diferenças no consumo de memória e energia. A Seção 5.9 conclui o capítulo.

5.2 Processo de Autenticação

O processo de associação de um nó na rede é feito em três vias [1], e pode ser visto resumidamente na Figura 5.1.

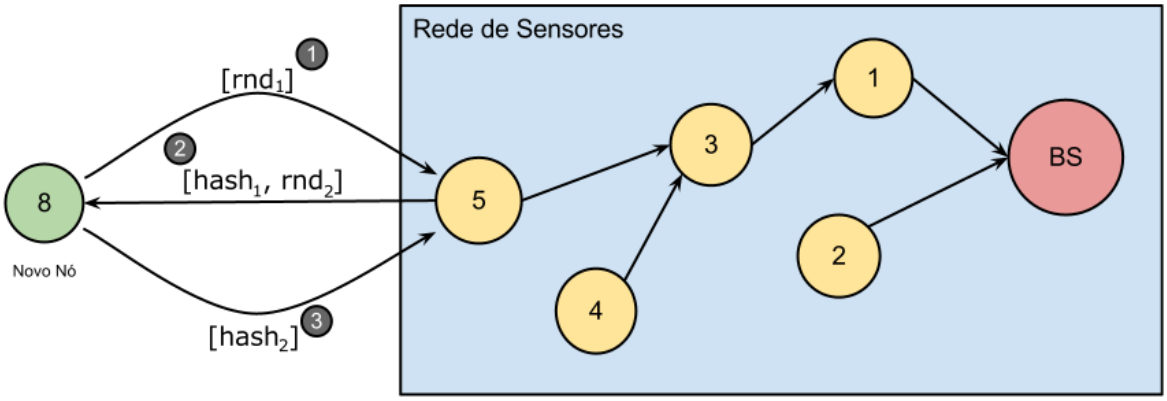


Figura 5.1: Processo Autenticação em três vias (*3-Way Handshake*)

Durante o processo é utilizado um chamado código de confirmação. Esse código é obtido a partir da chave pré-compartilhada e de um valor aleatório conhecido pelo outro dispositivo ao qual deseja-se autenticar. Da seguinte função descreve o algoritmo para a geração do código de autenticação:

$$hash_n = Fn(Fn(psk) + rnd_n)$$

Onde:

hash Representa o código de autenticação referente ao número aleatório enviado,

psk Representa a chave pré-compartilhada única em toda a rede,

rnd Representa um número aleatório, recebido do nó a quem está sendo feito a autenticação,

Fn Representa uma função de *hash* segura.

A definição da função *hash* a ser utilizada foge o escopo desse trabalho, ficando ao encargo do leitor utilizar algoritmos de *hash* que melhor se adaptam a suas aplicações. Diversas imple-

[1] O termo comumente adotado e *three-way handshake* ou "aperto de mão em três vias"

mentações de algoritmos de *hash* são disponíveis para TinyOS(LIU PANOS KAMPANAKIS, 2012; OLIVEIRA et al., 2012; AVRAES. . . , 2012).

5.2.1 Processo de Supervisão

Durante o processo de autenticação, tem-se o denominado processo de supervisão. O processo de supervisão consiste em um contador que limita o tempo máximo de resposta a uma determinada chamada. Caso esse tempo atinja o limite permitido, o estado atual de autenticação será desconsiderado e qualquer mensagem ao respectivo estado será descartado. Esse processo é utilizado como um dos mecanismos que visam mitigar possíveis ataques ao modelo proposto. Alguns desses ataques podem ser analisados na Seção 5.4.

5.2.2 Mensagem 1 - Requisição (*MODAUTH_MSG1*)

Envio

Quando um nó deseja ingressar na rede, é enviada uma mensagem responsável por informar aos nós da rede ao alcance que um novo nó deseja autenticar-se. Essa mensagem é transmitida para todos os nós (*broadcast*) ao alcance e possui em um de seus campos um código randômico gerado pelo nó de origem. Esse código é utilizado para garantir que ele só aceite respostas de nós autênticos da rede, ou seja, de um nó que compartilhe a mesma chave que o nó que enviou a mensagem.

Recepção

Ao receber uma mensagem de requisição de autenticação, um nó legítimo à rede gera um *hash* obtido através da chave pré-compartilhada e o valor aleatório contido na mensagem. Esse *hash* será enviado como mensagem de aceitação, a fim de que o nó transmissor da mensagem original concorde ou discorde que ambos possuem a mesma chave pré-compartilhada.

É imprescindível que, ao receber uma mensagem requisição, o nó verifique se a mensagem foi enviada para todos os nós ou se foi endereçada a ele, aceitando somente as mensagens que

forem enviadas em *broadcast*. Esse procedimento é necessário para evitar alguns cenários de ataques vislumbrados na Seção 5.4.

5.2.3 Mensagem 2 - Aceitação (*MODAUTH_MSG2*)

Envio

A mensagem de aceitação é gerada a partir da mensagem de requisição inicial. Ela é utilizada para que o nó que solicitou a requisição de autenticação saiba que o nó que a recebeu compartilha da mesma chave pré-compartilhada. Ai enviar uma mensagem de aceitação, o nó inicia o processo de supervisão. A supervisão consiste em cronometrar um tempo limite máximo de aceitação de resposta da mensagem. Caso o tempo expire a supervisão será

Recepção

Ao receber a mensagem de requisição de autenticação de um nó qualquer, os nós irão enviar uma mensagem de retorno contendo um código de autenticação referente ao número aleatório recebido, juntamente com outro código aleatório que será utilizado na autenticação do nó que deseja autenticar-se à rede. Após enviar a mensagem de Aceitação, o nó quem gerou o valor aleatório iniciará um método de supervisão que aguardará o código de Confirmação (Seção 5.2.4) por um período de tempo determinado.

O processo de supervisão consiste em um parâmetro utilizado para definir por quanto tempo o valor aleatório gerado será válido. Isso garante a eficiência da autenticação perante ataques de força bruta, onde um atacante solicita o ingresso na rede e tenta múltiplas respostas a fim de acertar alguma e conseguir com sucesso o ingresso na rede.

Ao receber a mensagem de aceitação, o nó irá verificar se o código de autorização realmente é equivalente à sua chave pré-compartilhada e ao número aleatório que foi enviado. Caso positivo, o nó passará a confiar no nó que lhe enviou a mensagem, pois o mesmo possui sua mesma chave chave pré-compartilhada.

5.2.4 Mensagem 3 - Confirmação (*MODAUTH_MSG3*)

Após o recebimento do código de aceitação, o nó que deseja autenticar-se a rede valida a confiabilidade do nó que está enviando a mensagem através do cálculo do código de autenticação. Verificado a confiabilidade do nó vizinho, o nó corrente necessita informar ao nó vizinho que ele também é confiável. Para isso ele envia um código de autenticação referente ao valor aleatório enviado pelo código vizinho na mensagem de aceitação.

Ao receber uma mensagem de confirmação, o nó verifica se existe alguma supervisão não expirada que autentique o nó que enviou a mensagem. Feito a verificação e garantida a confiabilidade, o processo de autenticação é concluído e os nós passam a comunicar-se entre si.

5.3 Proteção da Chave Pré-compartilhada(PSK)

Um dos problemas da abordagem proposta está no acesso físico aos nós sensores, onde dependendo da aplicação os nós ficariam exposto a ambientes públicos e sujeitos a ataques físicos. Com isso, um atacante poderia comprometer um nó sensor, remover a memória de programa (ROM) e então ler a chave pré-compartilhada, o que comprometeria toda a rede, pois seria possível implementar um nó que se utilize a mesma chave e, consequentemente, passe a ser confiável para rede.

A abordagem proposta para a proteção da chave pré-compartilhada é não armazena-la em qualquer memória não volátil (como a memória *ROM*). A melhor alternativa seria armazenar a chave na memória RAM, pois ao cortar a alimentação (morte do nó) ou ao remover fisicamente a memória RAM em uma tentativa de ataque, em segundos a chave se perderia e não seria possível recuperá-la.

Como a memória RAM é volátil, não é possível armazenar a chave diretamente nela sem termos que lê-la de um meio não volátil. Com isso, foi utilizado a interface de acesso a memória Flash disponível no TinyOS (GAY; HUI, 2012) para armazenar a chave, porém com a particularidade de não manter a chave armazenada na memória flash após a leitura.

Após o sensor ser ligado, ele coleta a estrutura de dados a chave pré-compartilhada na memória Flash utilizando o componente *ConfigStorageC* nativo do *TinyOS*. Essa estrutura contém o tamanho da chave, a soma verificadora da chave, utilizada para garantir a integridade da chave, e a chave propriamente dita. Após o sucesso da leitura, o módulo posta uma tarefa que será responsável por sobrescrever a chave da memória Flash, fazendo com que, a partir desse momento a chave só esteja disponível na memória RAM e que, ao perder a alimentação, seja por falta de energia ou por comprometimento do nó, a chave seja perdida.

É importante ressaltar que, mesmo após a perda da corrente elétrica, a memória *RAM* continua com os dados armazenados por alguns segundos. Em algum casos, é possível efetuar o chamado *Cold Attack*(HALDERMAN et al., 2009) no dispositivo, onde a memória é congelada e com isso ela mantém-se energizada por até 10 minutos. Porém esse último ataque requer uma intervenção preparada ao dispositivo sensor e, além disso, existe o tempo de remoção da memória, blindagens do nó sensor entre outros aspectos que dificultariam o sucesso do ataque.

Assim como não é possível gravar um dado na memória RAM sem telo em uma memória não volátil, não é possível armazenar a chave pré-compartilhada na memória Flash sem ser pelo próprio programa. Porém, se fosse utilizado o próprio ModAuth para armazenar a chave, ela ficaria também armazenada na memória ROM. Para isso foi criado o *PSKWriter*, que é uma aplicação criada somente para a gravação da chave pré-compartilhada na memória Flash. Portanto, o processo de instalação de uma aplicação que utilize o ModAuth consiste na gravação do PSKWriter que, ao ser executado, irá escrever a chave pré-compartilhada na memória Flash e, em seguida, a gravação do programa propriamente dito.

5.4 Validação Informal

Como a proposta possui somente 3 transações, não foi feito uma avaliação formal de validação. Portanto, dado a simplicidade, a validação foi feita de maneira empírica, analisando as possíveis tentativas de ataques em que o modelo proposto estaria sujeito.

5.4.1 Ataque de Interceptação e Geração de Mensagens

Is seguintes métodos de ataque foram avaliados em cada uma das mensagens a fim de obter a chave pré-compartilhada ou se passar por um nó confiável para a rede:

1. Interceptação da mensagem (Figura 5.2)

2. Envio da mensagem

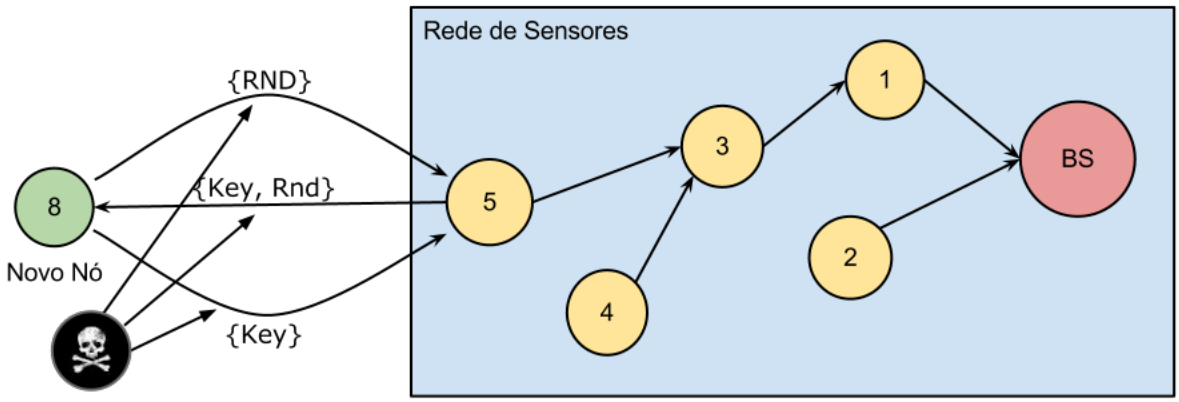


Figura 5.2: Interceptação do Processo Autenticação em três vias (*3-Way Handshake*)

Como os dados são trafegados em um ambiente não seguro, para garantir que o modelo proposto garanta a confiabilidade da vizinhança, foi analisado teoricamente diversos cenários de ataques que envolvam ambos, a leitura das mensagens do processo de autenticação e a geração de mensagens falsas tanto no começo quanto no meio do processo de autenticação.

Um atacante pode interceptar a mensagem referente à primeira fase de autenticação, porém essa mensagem não possui qualquer informação sigilosa e é uma mensagem puramente aberta, onde até o próprio atacante pode gerar tal mensagem e transmitir.

A segunda fase do processo originalmente é transmitida por um nó autêntico à rede. Apesar de ser possível obter o primeiro valor aleatório e seu respectivo código de autenticação, garantindo que estejamos trabalhando com uma função *hash* segura, não seria possível obter o valor da chave. Se um nó maléfico porém, a fim de tentar identificar a chave pré-compartilhada, responder a uma tentativa de autenticação de um nó legítimo, este não aceitará a resposta, pois o *hash* de autenticação não será correspondente ao valor que foi enviado.

A terceira fase do processo pode ser tratada semelhante à segunda fase, ou seja, um nó ao interceptar o código de autenticação não conseguirá obter a chave pré-compartilhada e, se um dado nó enviar um código de autenticação não será validado por um nó autêntico na rede.

5.4.2 Ataque *Woman-in-the-middle*

Um cenário um pouco mais complexo vista tentar autenticar-se um nó não autentico na rede sem que ele sequer conheça a chave pré-compartilhada, utilizando outro nó autêntico na rede para a geração. Esse cenário pode ser vislumbrado na Figura 5.3 e necessita de, pelo menos, dois nós autênticos próximos ao nó maléfico.

O nome *woman-in-the-middle* deu-se pelo fato de a abordagem utilizada no ataque não se enquadrar em nenhuma técnica já conhecida (*mitm*, *spoofing*, *replay attack* etc). Com isso, a técnica foi nomeada de *woman-in-the-middle*, por mais se aproxima da técnica *man-in-the-middle*, analisada na Seção 5.4.3.

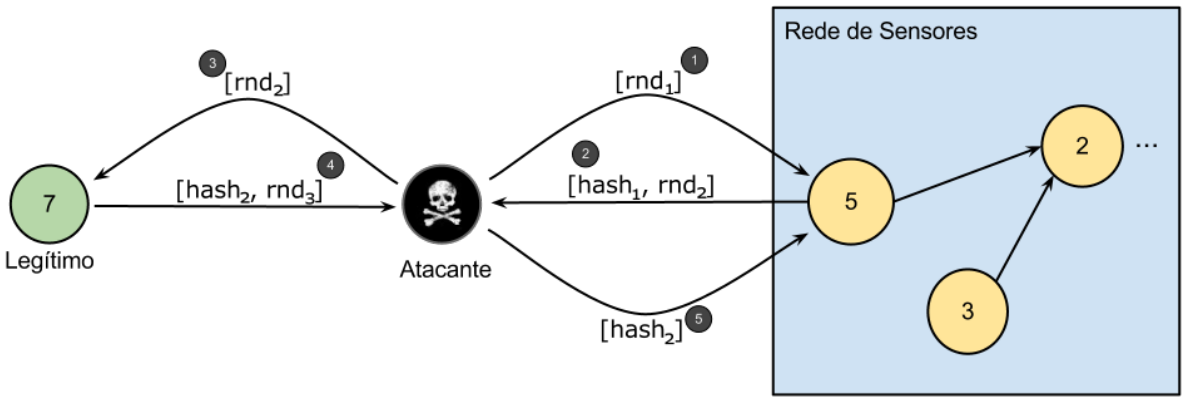


Figura 5.3: Interceptação do Processo Autenticação em três vias (*3-Way Handshake*)

Cada um dos passos podem ser analisados abaixo:

1. O primeiro passo segue normalmente, onde o nó maléfico envia um código aleatório rnd_1 para a rada a fim de autenticar-se a ela,

2. Sem saber se o nó é autentico ou não, os nós da rede respondem com o código de autenticação key_1 referente o valor rnd_1, esperando que o outro nó seja autêntico e já passe a confiar nele. Além disto, é enviado um valor rnd_2, para que o nó que iniciou o processo

de autenticação para que ele responda com o respectivo código de autenticação key_2 e ingresse na rede,

3. Em posse do valor aleatório rnd_1 e sem saber como gerar a chave pré-compartilhada válida referente à ele, o nó maléfico retransmite o valor rnd_1 para um outro nó autêntico na rede, a fim de "iniciar" um novo processo de autenticação,

4. Um dado nó legítimo da rede, sem saber malevolência do nó que enviou o valor rnd_2, transmite a chave key_2 para ele juntamente com um valor rnd_3, a fim de iniciar um processo de autenticação,

5. O nó maléfico, mesmo sem saber como foi gerado, agora está em posse do código de autenticação key_2 referente ao valor rnd_2 e então o transmite ao nó que inicialmente gerou o valor rmd_2, que passa a confiar nele.

Apesar da lógica está correta, algumas características de implementação inibem a tentativa de ataque:

- A primeira mensagem obrigatoriamente deve ser enviada em *broadcast*. Quando um nó sensor legitimo receber uma mensagem inicial de requisição, ele verifica se a mesma foi enviada em *broadcast*. Caso a mensagem tenha sido endereçada ao nó, será considerado uma situação atípica do funcionamento normal da rede e a mensagem será descartada. Com isso, a mensagem do passo 3 da Figura 5.3 também será recebida pelo nó 5 que gerou o valor rnd_1 e, portanto, poderá cancelar qualquer supervisão referente ao valor solicitado.

- Um dos parâmetros é o **MODAUTH_TIME_WAIT** (veja Seção 5.5). Esse parâmetro pode ser utilizado para, caso ocorra uma tentativa de ataque, o tempo já tenha expirado e o nó não aceite mais a resposta da requisição em questão. Durante o trabalho não foi avaliado qual seria o valor mais recomendado para o parâmetro **MODAUTH_TIME_WAIT**, de forma que iniba a rede de efetuar o

- Os nós 7 e 5 são idênticos na rede, possuindo as mesmas chaves pré-compartilhadas e os mesmos parâmetros de configuração. Com isso, a requisição 1 foi recebida tanto pelo nó 5 quanto pelo nó 7 e ambos enviaram sua resposta (mesmo que não tenha sido atendidas todas). Uma forma de evitar esse ataque seria não processar qualquer mensagem de requisição quando alguma supervisão para um determinado nó estiver sendo executada.

5.4.3 Ataque *Man-in-The-Middle*

O ataque *man-in-the-middle* (MiTM) é muito comum nas redes de computadores *Ethernet* cabeada. O mesmo ataque foi avaliado no modelo de autenticação e, para que ele faça sentido e não seja impedido pelos procedimentos inibidores citados anteriormente, ele precisa ser aplicado em um cenário estritamente específicos.

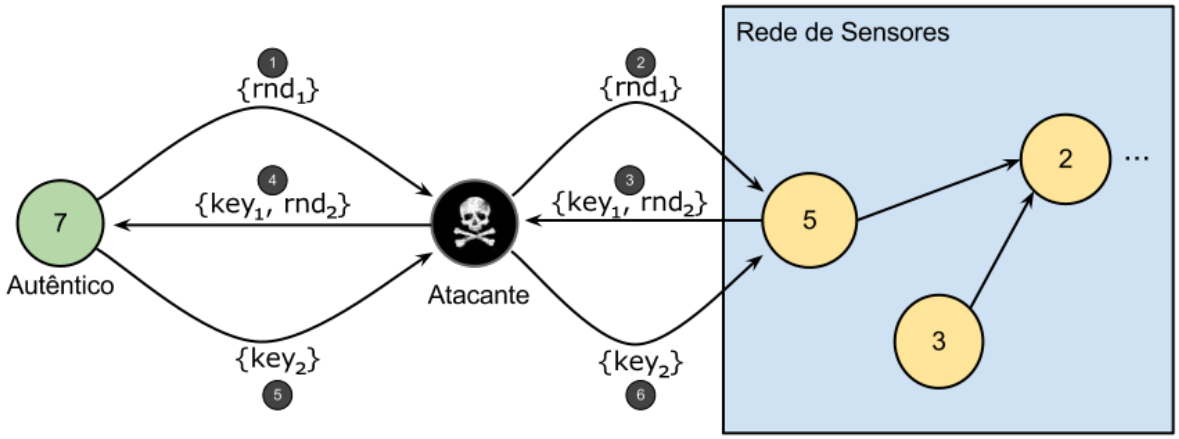


Figura 5.4: Ataque *MiTM* no Processo de Autenticação)

O procedimento do ataque MiTM pode ser visto na Figura 5.4, e as seguintes características fazem parte deste cenário:

- Os nós 5 e 7 devem ser legítimos para a rede e possuírem a mesma chave pré-compartilhada,
- Os nós 5 e 7 não podem conseguir comunicar um com o outro,
- O nó **atacante** deve estar mais próximo do nó 5, de forma que seja possível transmitir a mensagem 2 para o nó e o nó 7 também não receba

- É necessário que o nó 7 ainda não esteja autenticado à rede, pois a primeira mensagem a ser analisada é uma solicitação de requisição partida do nó 7.

Para evitar o sucesso nesse ataque, basta definir parâmetro de configuração referente ao tempo limite de supervisão (Seção 5.5) que, com a execução do ataque, o tempo utilizando representa o dobro do tempo do envio de uma mensagem normal na rede.

5.5 Parâmetros de Configuração

Diversos parâmetros tornam o módulo de autenticação ModAuth flexíveis a aplicações distintas. Alguns parâmetros são passíveis de alteração simplesmente no arquivo de compilação *Makefile* e outro devem ser alterados diretamente no código, pois sua alteração necessitará de ajustes no código a ser utilizado.

Para evitar conflito com outros parâmetros da aplicação, todos os parâmetros da proposta são prefixados de *MODAUTH_*. Eis suas definições:

MODAUTH_PSK Esse parâmetro representa a chave pré-compartilhada propriamente dita. Ele é possui o valor padrão *this_is_just_a_test_key* e DEVE ser redefinido no arquivo *Makefile*. Caso não seja redefinido um alerta proposital irá aparecer durante o processo de compilação.

MODAUTH_MAX_KEY_LEN Esse parâmetro representa o tamanho máximo que a chave pode ter. Alterar esse tamanho implica no desempenho de memória de dados e de programa.

MODAUTH_HASH_LEN Esse parâmetro indica o tamanho em bytes do *hash* utilizando no processo de autenticação. Seu valor padrão é 16 bytes e ele não pode ser alterado através do *Makefile*, pois depende da alteração da função *hash* utilizada.

MODAUTH_MAX_NEIGH Esse parâmetro representa o número de vizinhos máximo que um nó pode ter. Seu valor padrão é 5 e pode ser alterado no *Makefile*. A alteração desse

parâmetro implicará drasticamente no desempenho e na escalabilidade da aplicação. Seu parâmetro deve ser definir de acordo com a organização esparsa ou densa que a aplicação final se propõe a trabalhar.

MODAUTH_PSK_STORAGE_ADDR Esse parâmetro representa o endereço na memória Flash onde será armazenado a chave pré-compartilhada. Caso a aplicação trabalho com memória Flash esse parâmetro é utilizado para evitar conflitos de endereçamento. Seu valor padrão é 0x00 e pode ser alterado pela *Makefile*.

MODAUTH_TIME_WAIT Esse parâmetro representa o tempo que a aplicação ficaria esperando para ter a resposta no processo de autenticação. Sua unidade é segundo, seu padrão é 2 segundos, e ele pode ser alterado pelo *Makefile*.

5.6 Estruturas Internas

A primeira estrutura a ser apresenta é a estrutura de armazenamento da chave pré-compartilhada. Essa estrutura pode ser vista na Figura 5.5 e seus campos possuem os seguintes significados:

Soma Verificadora Um cálculo de integridade que garante que a chave realmente é válida e não foi alterada. Esse código é utilizado para verificar se a chave armazenada não foi modificada desde sua gravação.

Tamanho O tamanho da chave armazenada em bytes.

PSK A chave propriamente dita.

Como dito na Seção 5.3, essa estrutura é armazenado na memória Flash pela aplicação *PSKWrite* que faz parte do módulo *TinyAuth*. Essa aplicação obtém o valor da chave do arquivo de compilação *Makefile*, calcula o tamanho e a soma verificadora e, em seguida, armazena a chave na memória *Flash* do nó sensor.

A próxima estrutura a ser definida é a responsável por armazenar informações da vizinhança, cujo formato está na Figura 5.6. Essa estrutura representa o item de um vizinho, que

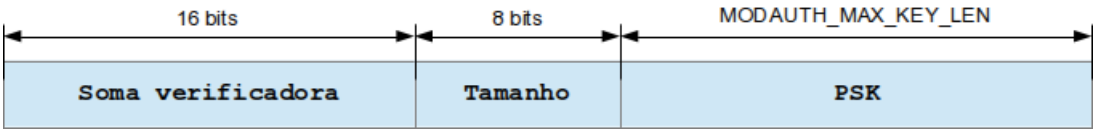


Figura 5.5: Estrutura da chave pré-compartilhada armazenada na memória Flash

faz parte de uma lista estática de vizinhos de tamanho definido pelo parâmetro de configuração *MODAUTH_MAX_NEIGH*(Veja Seção 5.5).

Segue a descrição de seus campos:

Ativo Representa seu o campo atual está ativo (alocado para um determinado vizinho) ou não. Um vizinho ativo não é necessariamente um vizinho confiável. Para isso utiliza-se o campo **Confiável** a seguir.

Confiável Representa se um determinado vizinho é confiável. Um vizinho passa a ser confiável se ele passar por todo o processo de autenticação descrito na proposta, possibilitando ao nó enviar e receber mensagens deste.

Ident. Vizinho Este campo armazena o código identificador do vizinho (*NODE_ID*) utilizado para enviar e receber mensagens.

Tempo Este campo representa o número de segundo que se passaram deste que o vizinho tentou a tentativa de autenticação. Ele é utilizando para definir o tempo limite de esperar pelo término processo. Esse tempo é definido pelo parâmetro **MODAUTH_TIME_WAIT**(Veja Seção 5.5)

Hash de Autenticação Este campo armazena o Hash de autenticação que se espera do vizinho. Esse valor foi pré-calculado para que não seja necessário recalculá-lo a cada mensagem de tentativas de autenticação falha (por nós não autênticos).

A Figura 5.7 representa a estrutura utilizada na comunicação durante o processo de autenticação. Essa mensagem é definida pelos seguintes campos:

Tipo Representa o tipo da mensagem, tendo os valores 0, 1 e 2 para cada um dos passos do processo de autenticação.

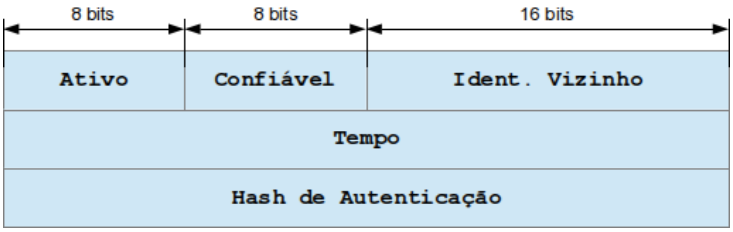


Figura 5.6: Estrutura dos elementos da lista de vizinhança

Random Um valor aleatório gerado em tempo real pelo nó que está transmitido a mensagem.

Hash de Autenticação O hash de autenticação da última mensagem recebida.

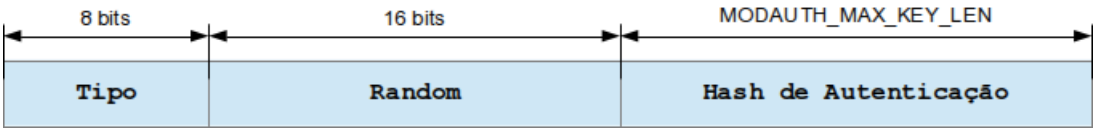


Figura 5.7: Estrutura da mensagem utilizada pelo módulo de autenticação

5.7 Interface do Componente TinyOS

A Figura 5.8 representa a interface de componentes TinyOS gerada pelo *nesdoc*.

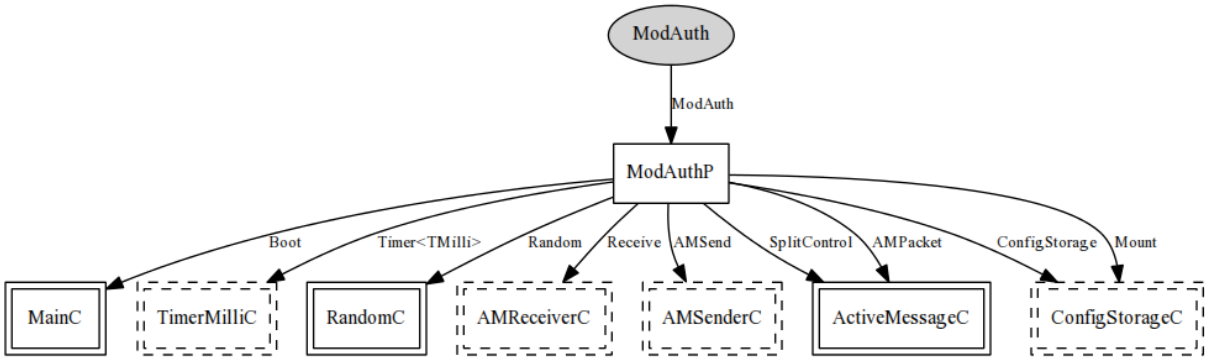


Figura 5.8: Interface de conexão dos componentes do *ModAuth*

A ligação com o componente *MainC* é fundamental para, no momento da inicialização do dispositivo sensor, o nó automaticamente iniciar o processo de autenticação a alguma rede próxima. Porém antes ele efetua o processo que obtém e apaga a chave pré-compartilhada da memória Flash (Seção 5.3) para, então, iniciar o processo de autenticação.

O componente *MainC* foi utilizado para acesso ao evento *MainC.booted*, onde o módulo fará a leitura da chave e iniciará o processo de autenticação. O componente *TimerMillicC* é o

contador utilizado para dois propósitos: o primeiro consiste em gerar a tarefa responsável por enviar as requisições de solicitação de acesso à rede caso o nó atual ainda não esteja autenticado a qualquer nó legítimo. Sua segunda função é cronometrar o tempo de supervisão das requisições de autenticação pendentes e, caso expirado, cancelá-los. O componente *RandomC* é responsável por gerar números pseudo-aleatórios, utilizados no processo de autenticação. Os componentes *AMReceiverC*, *AMSenderC* e *ActiveMessageC* são responsáveis pelo processo de envio, recepção e configuração do rádio, respectivamente. O componente *ConfigStorage* é responsável por todo o acesso, leitura e escrita da memória *Flash*.

5.8 Avaliação Experimental

A avaliação experimental foi feita utilizando o simulador Avrora e como dados foram coletados os consumos de energias de cada um dos nós. Os fatores de avaliação definidos foram:

1. Tamanho da memória RAM/ROM
2. Consumo de energia da CPU
3. Consumo de energia do rádio
4. Consumo de energia de acesso à memória *flash*
5. Consumo de energia acumulado (CPU + rádio + flash)

As seguintes características compõem a experimentação:

- Como fator de variação foi estabelecido o número de vizinhos que um nó sensor pode ter, analisando o resultado dos dados a serem coletados para uma vizinhança que varia de 1 e 10 nós.
- A avaliação foi feita em uma simulação de 5 minutos, e os dados apresentados nos resultados representam a média aritmética de toda uma vizinhança.

- Como programas de teste foram utilizando duas versões do aplicativo *RadioCountToLeds*[2], sendo uma delas a padrão e a outra utilizando o módulo de autenticação TinyAuth.

5.8.1 Topologia de Avaliação

A topologia de avaliação pode ser vista na Figura 5.9. O alcance do rádio padrão no simulador *Avrora* são 15 unidades[3], portanto na topologia proposta existem 10 grupos de redes onde um nó consegue alcançar qualquer outro nó em sua mesma coordenada vertical e não consegue alcançar qualquer outro nó da rede. Com isso temos 10 redes isoladas e com tamanho de vizinhança diferentes para fazer a análise e coletar o consumo de energia.

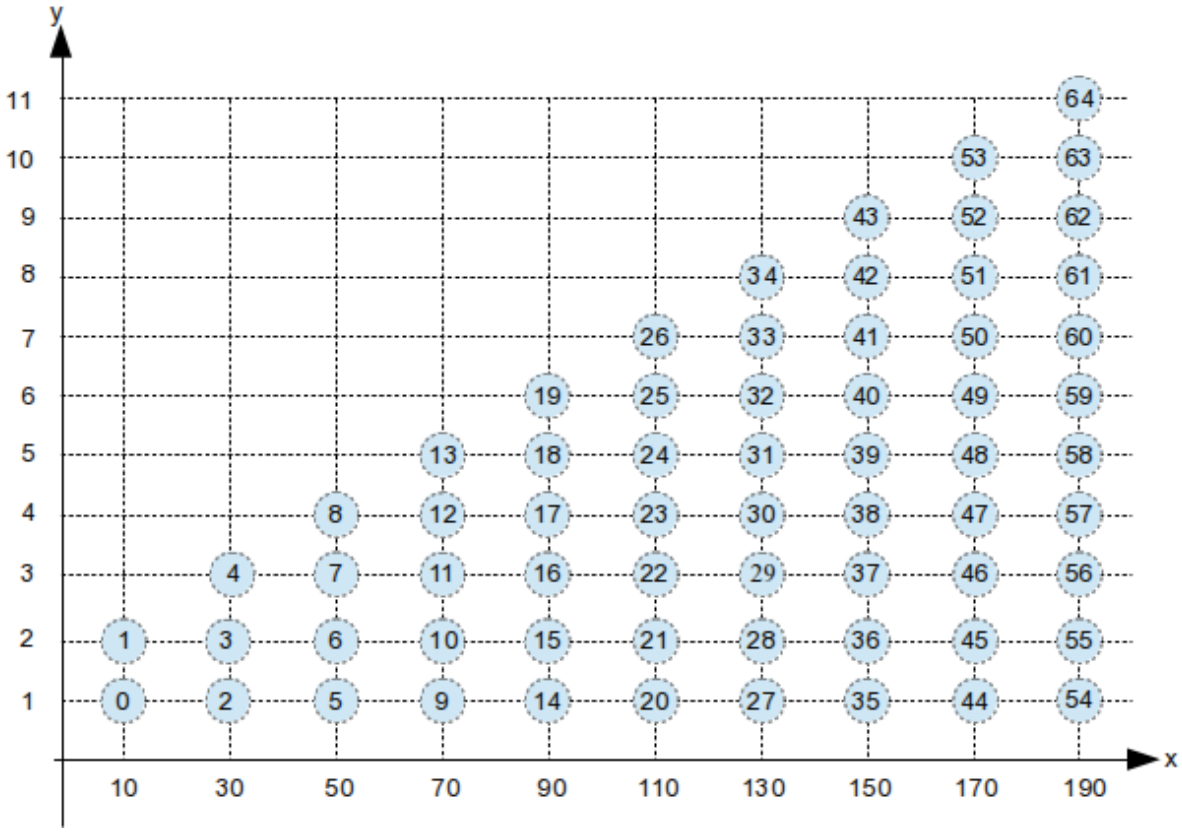


Figura 5.9: Toppologia utilizada para avaliação da proposta

[2]Disponível em *$TOSROOT/apps/RadioCountToLeds/*
[3]Não é documentado o que essa unidade representa

5.8.2 Consumo de Memória RAM/ROM

A primeira grandeza a ser avaliada é o consumo de memória RAM e memória ROM. A Tabela 5.1 expressa a diferença entre os consumos de memória. Repare que apesar de o módulo ModAuth ter consumido 9724 bytes de memória a mais no aplicação (aproximadamente 9.5K bytes), esse valor é considerado pequeno comparado aos 128K bytes disponíveis nos dispositivos Micaz. O mesmo vale para a utilização da memória de dados (memória RAM) que teve um acréscimo de 465 bytes em uma capacidade máxima de 4K bytes. Na Figura 5.10 ambos os consumos de memória ROM e RAM podem ser vistos em proporção entre a aplicação sem o ModAuth, com o ModAuth e total de memória disponível.

	Memória ROM	Memória RAM
Sem ModAuth	11618	311
Com ModAuth	21342	776

Tabela 5.1: Utilização de memória (em bytes)

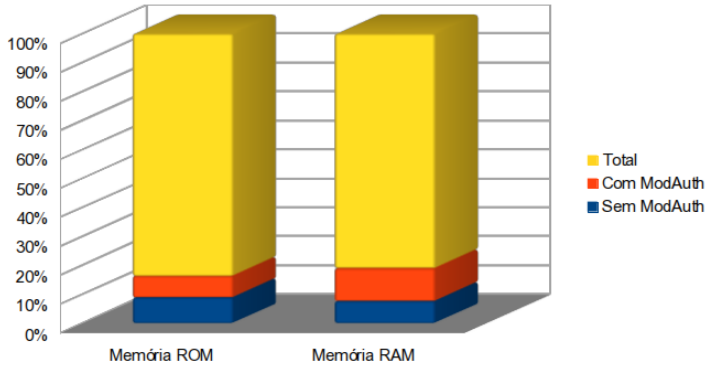


Figura 5.10: Gráfico de consumo de memória ROM/RAM

5.8.3 Consumo de Energia

Com excessão do acesso à memória Flash, a diferença de consumo de CPU e de Rádio foi irrisória, variando somente em alguns centésimo e milésimos de Joules, respectivamente.

O consumo de energia médio de CPU para cada uma das redes pode ser visto na Figura 5.11. Com o aumento da vizinhança, o consumo de energia aumenta de maneira gradativa em ambas as aplicações, com e sem o módulo *ModAuth*. O consumo da rede com o *ModAuth* é

maior pois a verificação de autenticação é feito antes do envio de cada um dos comandos. Essa verificação simplesmente testa se o nó atual está associado a algum outro nó da rede, e não propõe isso como um meio de segurança, pois um nó maléfico pode enviar dados sem fazer essa verificação, cabendo aos demais nós autênticos não confiarem nos dados transmitidos.

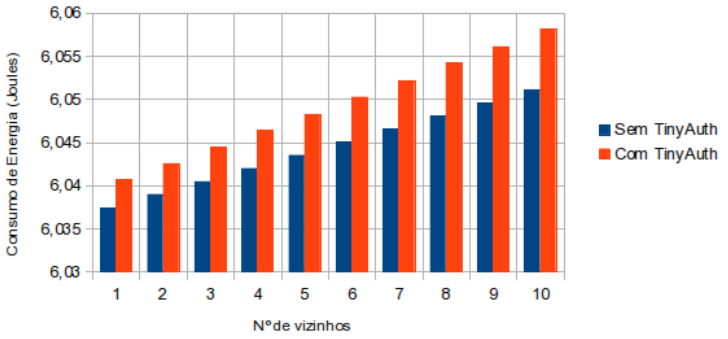


Figura 5.11: Gráfico de consumo de energia (CPU)

Na Figura 5.12 é possível analisar o consumo de energia médio do rádio de cada uma das redes. É interessante ressaltar que o decréscimo de consumo de energia dá-se pelo tempo de inicialização do simulador para cada uma das redes, pois para simular um ambiente mais próximo do real, foi inserido um intervalo de tempo entre a inicialização de cada um dos nós, de modo que eles não tenham a execução completamente sincronizadas.

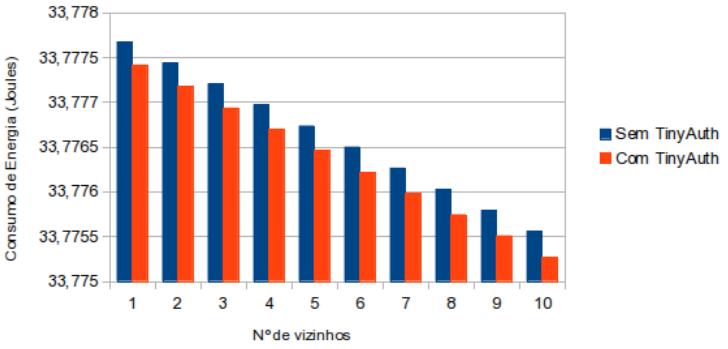


Figura 5.12: Gráfico de consumo de energia (Radio)

As Figuras 5.11 e 5.12 demonstram que, enquanto a proposta apresenta um acréscimo de consumo de energia de CPU, a mesma consome menos energia de rádio do que a aplicação sem o módulo de autenticação (a justificativa foi explicada nos parágrafos anteriores). A Figura 5.13 apresenta o consumo de energia de ambos CPU e rádio somados e mostra que, mesmo o consumo ter melhorado em um e piorado no outro, a utilização do módulo *ModAuth* proporciona

um consumo maior de energia de rádio de CPU.

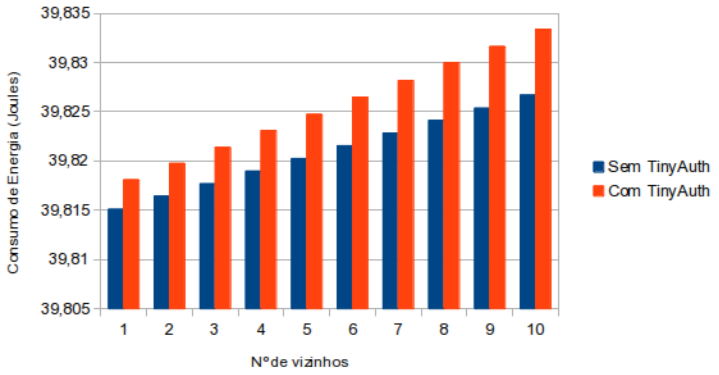


Figura 5.13: Gráfico de consumo de energia (CPU + Radio)

O grande consumo de energia foi devido ao acesso à memória Flash que, em comparação com alguns centésimos de Joules do consumo de energia e processamento, consumiu mais de 7 Joules de diferença em comparação com a aplicação sem o módulo de autenticação [4]. O resultado dessa diferença pode ser visto na Figura 5.14.

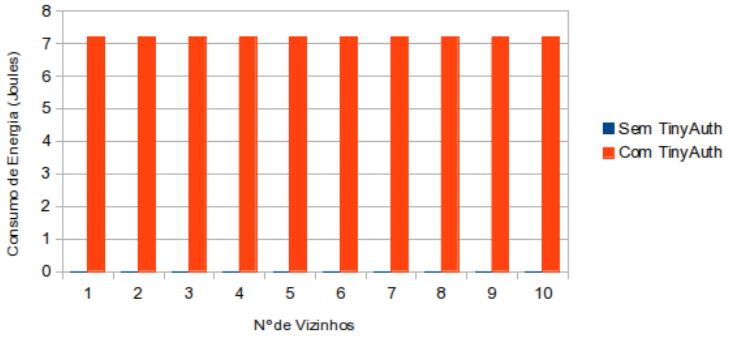


Figura 5.14: Gráfico de consumo de energia (Flash)

A Figura 5.15 apresenta o consumo total de energia do rádio, processamento e utilização da memória flash.

5.9 Conclusões do Capítulo

Este capítulo apresentou o *TinyAuth*, um módulo de autenticação para redes de sensores baseado em chave pré-compartilhada. Com esse módulo foi possível definir o processo de autenticação necessário para garantir a autenticidade da vizinhança. Além disto, foi proposta um

[4] A aplicação normal não faz qualquer uso de memória *Flash* em sua implementação

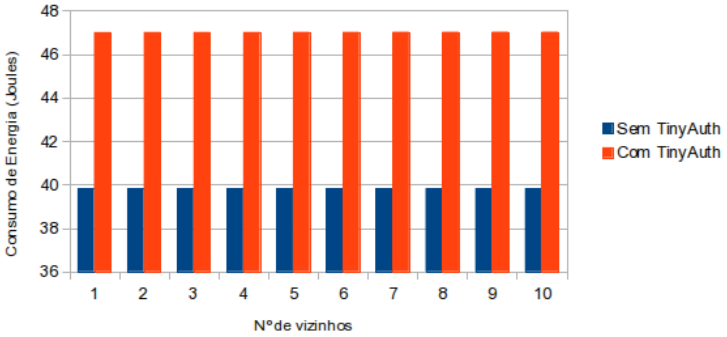


Figura 5.15: Gráfico de consumo de energia (Total)

mecanismo que visa dificultar a descoberta da chave em casos onde o nó seja comprometido ou morra[5]. Foram definidos os parâmetros de configuração que permitem garantir a escalabilidade do nó em aplicações diversas. A validação do protocolo foi feito de maneira empírica, dado a sua simplicidade, avaliando as possíveis condições de ataque que o modelo estaria sujeito. Foi apresentado a interface de componentes de programação TinyOS para o módulo proposto e, como avaliação experimental, foi mensurado o consumo de memória e energia de uma aplicação exemplo com e sem o módulo de autenticação.

[5] A morte de um nó dá-se pelo término de sua energia

6 Conclusões

Assim como as redes de computadores convencionais, as Redes de Sensores Sem Fio estão sujeito aos mais diversos ataques, como negação de serviço, ataque de falsificação de endereço, e ataques específicos para redes de sensores. Com a perspectiva de utilização massiça de (redes de) sensores monitorando os mais diversos aspectos do cotidiano, particularmente em um cenário de Internet das Coisas e Cidade Inteligentes, a preocupação com a segurança toma uma dimensão ainda maior e mais preocupante, considerando que dados críticos passarão a circular a todo o instante pela infra-estrutura de comunicação provida pelas redes. Por exemplo, com a instrumentaçao das cidades com redes de sensores, torna-se viável o desenvolvimento de soluções que visam monitorar grandezas urbanas a fim de promover uma melhora na gestão pública. Com tais sensores distribuídos em locais públicos, em áreas críticas, áreas de de extensa cobertura geográfica, e com os dados trafegando através da rede de comunicação, a infra-estrutura de coleta e disseminação de dados provida pelas RSSF torna-se suscetível aos mais diversos problemas de segurança. Neste sentido, a realização de pesquisas em segurança em RSSF se reveste de grande importância, e abre espaço para o desenvolvimento de trabalhos voltados para a criação de novos algoritmos, protocolos, ferramentas e ambientes específicos de RSSF que contribuam para o aumento da segurança da rede.

Nesta dissertação, foi realizado um trabalho de desenvolvimento de um mecanismo de segurança que permite garantir a autenticidade entre nós da rede, juntamente com um ambiente de experimentação de aplicações para redes de sensores. O módulo desenvolvido, baseado em chave pré-compartilhada, apresentou um acréscimo de energia no processamento e no rádio insignificante comparado à aplicação sem a utilização do módulo proposto.

Além disto, o trabalho propõe uma arquitetura de simulação que permite testar, comunicar e coletar resultados de redes IPv6 para baixo consumo (6LoWPAN). Essa arquitetura foi baseada no simulador Avrora e também possibilita o acesso de aplicações externas aos nós do ambiente simulado de maneira transparente. Com isso, permite a utilização da proposta como um ambiente de *testbed* remoto virtualizado.

Apesar da eficiência e do modelo de autenticação garantir a autenticidade, observa-se que ele não deve ser utilizado isoladamente para garantir a integridade ou qualquer outro pilar de segurança da informação.

Embora o trabalho desenvolvido esteja alinhado com os seus objetivos iniciais, uma série de melhorias e desdobramentos são vislumbrados, sendo estes fontes naturais de trabalhos futuros. Por exemplo, seria importante desenvolver e testar algumas aplicações remotas acessando a rede de sensores, a fim de vislumbrar mais de perto a possibilidade da criação de um *testbed* inteiramente (ou parcialmente) simulado. Além disso, seria também interessante conseguir promover um ambiente de simulação que permitisse simular outras plataformas de nós sensores além da plataforma MICAz usada no trabalho. Assim, seria possível simular, por exemplo, uma rede com nós TelosB e, consequentemente, testar o TinyOS em sua última versão do blip 2.0 com seu recém-lançado protocolo de roteamento RPL.

No trabalho desenvolvido, todo o ambiente de simulação é executado em modo texto e com a execução manual de todos os comandos. Logo, uma outra proposta interessante é a criação de uma interface gráfica que integre e abstraia a comunicação entre os processos, fornecendo uma forma amigável e de fácil utilização para os desenvolvedores de redes de sensores.

Outros trabalhos adicionais que também podem ser vislumbrados em continuação a essa pesquisa são listados abaixo:

1. A montagem de um ambiente de testes remoto onde fosse possível alocar recursos para simulação de redes gigantes em máquinas com alto poder computacional,
2. Integrar o trabalho com outros módulos de segurança em RSSF a fim de promover um mecanismo geral que garanta não somente a autenticidade dos nós, mas também os demais

pilares da segurança da informação,

3. A integração desta proposta a protocolos recentes da pilha 6LoWPAN, como os protocolos de roteamento RPL presentes no Blip 2.0 do TinyOS.

APÊNDICE A – Resultados Brutos

A.1 Consumo de Energia da aplicação *RadioCountToLeds* sem o Módulo TinyAuth

ID	CPU (Joules)	Radio	Flash	Total
0	6,0373625721	33,7779096538	0,0036	39,8188722259
1	6,0374785575	33,7774397461	0,0036	39,8185183036
2	6,0388441699	33,7779096538	0,0036	39,8203538237
3	6,0389839223	33,7774397461	0,0036	39,8200236683
4	6,0390585312	33,7769698689	0,0036	39,8196284001
5	6,0402919699	33,7779096538	0,0036	39,8218016236
6	6,0404247708	33,7774397461	0,0036	39,8214645169
7	6,0405223064	33,7769698689	0,0036	39,8210921753
8	6,0406592885	33,7764999918	0,0036	39,8207592803
9	6,0417527858	33,7779096538	0,0036	39,8232624395
10	6,0418906804	33,7774397461	0,0036	39,8229304265
11	6,0419915223	33,7769698689	0,0036	39,8225613913
12	6,0421076486	33,7764999918	0,0036	39,8222076405
13	6,0421788154	33,7760301147	0,0036	39,8218089301
14	6,0432114604	33,7779096538	0,0036	39,8247211142
15	6,0433444109	33,7774397461	0,0036	39,824384157
16	6,0434462891	33,7769698689	0,0036	39,824016158
17	6,0435803514	33,7764999918	0,0036	39,8236803433

18	6,0436630958	33,7760301147	0,0036	39,8232932105
19	6,0437791809	33,775560207	0,0036	39,8229393879
20	6,0447036906	33,7779096538	0,0036	39,8262133444
21	6,044834371	33,7774397461	0,0036	39,825874117
22	6,0449490314	33,7769698689	0,0036	39,8255189003
23	6,0450495193	33,7764999918	0,0036	39,8251495111
24	6,0451807633	33,7760301147	0,0036	39,824810878
25	6,0452886889	33,775560207	0,0036	39,8244488959
26	6,0453555216	33,7750903299	0,0036	39,8240458515
27	6,0461678232	33,7779096538	0,0036	39,827677477
28	6,0462935801	33,7774397461	0,0036	39,8273333261
29	6,0464035232	33,7769698689	0,0036	39,8269733921
30	6,0465244407	33,7764999918	0,0036	39,8266244325
31	6,046630653	33,7760301147	0,0036	39,8262607677
32	6,0467209158	33,775560207	0,0036	39,8258811228
33	6,046829668	33,7750903299	0,0036	39,8255199979
34	6,0469313124	33,7746204528	0,0036	39,8251517653
35	6,0476348292	33,7779096538	0,0036	39,829144483
36	6,0477623922	33,7774397461	0,0036	39,8288021382
37	6,047868998	33,7769698689	0,0036	39,8284388669
38	6,0479809002	33,7764999918	0,0036	39,828080892
39	6,048097707	33,7760301147	0,0036	39,8277278218
40	6,0481853663	33,775560207	0,0036	39,8273455733
41	6,0482969867	33,7750903299	0,0036	39,8269873166
42	6,0484054914	33,7746204528	0,0036	39,8266259443
43	6,0484928122	33,7741505757	0,0036	39,8262433879
44	6,0490540521	33,7779096538	0,0036	39,8305637059

45	6,0491968101	33,7774397461	0,0036	39,8302365562
46	6,0492864439	33,7769698689	0,0036	39,8298563129
47	6,0493910322	33,7764999918	0,0036	39,829491024
48	6,0495143212	33,7760301147	0,0036	39,829144436
49	6,0496076997	33,775560207	0,0036	39,8287679067
50	6,0497122845	33,7750903299	0,0036	39,8284026145
51	6,0498383868	33,7746204528	0,0036	39,8280588396
52	6,0499148551	33,7741505757	0,0036	39,8276654309
53	6,0500290258	33,773680668	0,0036	39,8273096938
54	6,0505716629	33,7779096538	0,0036	39,8320813167
55	6,0507010647	33,7774397461	0,0036	39,8317408107
56	6,0508164125	33,7769698689	0,0036	39,8313862815
57	6,050922941	33,7764999918	0,0036	39,8310229328
58	6,0510349601	33,7760301147	0,0036	39,8306650748
59	6,0511328805	33,775560207	0,0036	39,8302930876
60	6,0512354908	33,7750903299	0,0036	39,8299258207
61	6,0513352638	33,7746204528	0,0036	39,8295557166
62	6,0514467003	33,7741505757	0,0036	39,829197276
63	6,0515309741	33,773680668	0,0036	39,8288116421
64	6,0516283619	33,7732107909	0,0036	39,8284391528

A.2 Consumo de Energia da aplicação *RadioCountToLeds* Com o Módulo TinyAuth

ID	CPU (Joules)	Radio	Flash	Total
0	6,0408641825	33,7776482631	7,1869826074	47,005495053
1	6,0404764704	33,7771824961	7,1868826835	47,00454165

2	6,0426830508	33,777644153	7,1869826074	47,0073098112
3	6,0425701123	33,777178386	7,1868826835	47,0066311818
4	6,0423702999	33,7767043682	7,186782753	47,0058574211
5	6,0445081848	33,777644153	7,1869826074	47,0091349452
6	6,0444666846	33,7771701658	7,1868826835	47,0085195339
7	6,04442736	33,7767002581	7,186782753	47,0079103711
8	6,0442596906	33,7762344911	7,186682829	47,0071770107
9	6,0464051455	33,7776359328	7,1869826074	47,0110236857
10	6,0463875497	33,7771619456	7,1868826835	47,0104321787
11	6,046384288	33,7767002581	7,186782753	47,009867299
12	6,0464763243	33,7762221608	7,186682829	47,009381314
13	6,0463353982	33,7757522837	7,186582905	47,0086705869
14	6,0483025787	33,7776359328	7,1869826074	47,0129211189
15	6,0481953336	33,7771660557	7,1868826835	47,0122440728
16	6,0482282533	33,7766920379	7,186782753	47,0117030442
17	6,0483315252	33,7762262709	7,186682829	47,0112406251
18	6,0484162682	33,7757563938	7,186582905	47,010755567
19	6,0482353215	33,7752906268	7,1864829811	47,0100089294
20	6,0502538081	33,7776277126	7,1869826074	47,0148641281
21	6,0501894706	33,7771619456	7,1868826835	47,0142340997
22	6,0502209846	33,7766920379	7,186782753	47,0136957755
23	6,0502690031	33,7762139406	7,186682829	47,0131657727
24	6,050385956	33,7757481736	7,186582905	47,0127170346
25	6,0505484665	33,7752782965	7,1864829811	47,0123097441
26	6,0503811356	33,7748066752	7,1863826862	47,011570497
27	6,0521089695	33,7776277126	7,1869826074	47,0167192895
28	6,0520382238	33,7771537254	7,1868826835	47,0160746326

29	6,052063039	33,7766879278	7,186782753	47,0155337198
30	6,0521083027	33,7762180507	7,186682829	47,0150091824
31	6,0522528617	33,7757440635	7,186582905	47,0145798302
32	6,0524170753	33,7752741864	7,1864829811	47,0141742427
33	6,052446106	33,7748025651	7,1863826862	47,0136313573
34	6,0523701996	33,774332688	7,1862827622	47,0129856498
35	6,0540478239	33,7776236025	7,1869826074	47,0186540338
36	6,0540604481	33,7771496153	7,1868826835	47,0180927468
37	6,0540046121	33,7766879278	7,186782753	47,0174752928
38	6,0541037802	33,7762180507	7,186682829	47,0170046599
39	6,0542468011	33,7757440635	7,186582905	47,0165737696
40	6,0543128845	33,7752741864	7,1864829811	47,016070052
41	6,0544305317	33,7748025651	7,1863826862	47,015615783
42	6,0545391911	33,774332688	7,1862827622	47,0151546414
43	6,0544200729	33,7738628109	7,1861828382	47,0144657221
44	6,0558782011	33,7776236025	7,1869826074	47,020484411
45	6,0558128978	33,7771537254	7,1868826835	47,0198493067
46	6,0557993148	33,7766879278	7,186782753	47,0192699955
47	6,0559529234	33,7762139406	7,186682829	47,018849693
48	6,0560814092	33,7757440635	7,186582905	47,0184083777
49	6,0561437927	33,7752741864	7,1864829811	47,0179009601
50	6,056236989	33,7748025651	7,1863826862	47,0174222403
51	6,0563996765	33,774332688	7,1862827622	47,0170151267
52	6,0564289203	33,773866921	7,1861828382	47,0164786796
53	6,0562553478	33,7733929032	7,1860829078	47,0157311588
54	6,0577326923	33,7776112722	7,1869826074	47,0223265719
55	6,0577190989	33,7771413951	7,1868826835	47,0217431774

56	6,0577199135	33,7766755975	7,186782753	47,0211782639
57	6,0578706127	33,7762016103	7,186682829	47,020755052
58	6,0579875794	33,7757358433	7,186582905	47,0203063277
59	6,0581083027	33,7752659662	7,1864829811	47,0198572499
60	6,0581418857	33,7747943449	7,1863826862	47,0193189168
61	6,0583006138	33,7743244678	7,1862827622	47,0189078438
62	6,058364159	33,7738504806	7,1861828382	47,0183974778
63	6,058441545	33,773384683	7,1860829078	47,0179091358
64	6,0582872009	33,7729106958	7,1859829838	47,0171808804

Referências Bibliográficas

ASGAONKAR, A. Implementation of Attack on Harvard Architecture Devices by Code Injection. *International Journal of Computer Applications IJCA*, Foundation of Computer Science FCS, v. 7, n. 8, p. 38–43, 2010. Disponível em: <http://www.ijcaonline.org/volume7-/number8/pxc3871722.pdf>.

AVRAES: The AES block cipher on AVR controllers. Agosto 2012. Disponível em: <http://point-at-infinity.org/avraes/>.

AVRORA - The AVR Simulation and Analysis Framework. Junho 2012. Disponível em: <http://avrora.sourceforge.net/>.

AVRORA Sourceforge Project. Agosto 2012. Disponível em: <http://avrora.sourceforge.net/>.

CONTIKI OS. [S.l.]. Disponível em: <http://www.contiki-os.org/>. Acesso em: 20 de Junho de 2012.

DARGIE, W.; POELLABAUER, C. *Fundamentals of Wireless Sensor Networks: Theory and Practice*. [S.l.]: John Wiley & Sons, 2010. 336 p. ISBN 0470997656.

FRANCILLON, A.; CASTELLUCCIA, C. Code injection attacks on harvard-architecture devices. *Proceedings of the 15th ACM conference on Computer and communications security - CCS '08*, ACM Press, New York, New York, USA, p. 15, 2008. Disponível em: <http://portal.acm.org/citation.cfm?doid=1455770.1455775>.

FRASER, B. *Site Security Handbook*. IETF, set. 1997. RFC 2196 (Informational). (Request for Comments, 2196). Disponível em: <http://www.ietf.org/rfc/rfc2196.txt>.

GAY, D.; HUI, J. *Permanent Data Storage (Flash)*. [S.l.], 2012. Disponível em: <http://www-.tinyos.net/tinyos-2.x/doc/html/tep103.html>. Acesso em: 10 de agosto de 2012.

GAY, D. et al. The nesC language: A holistic approach to networked embedded systems. In: *Acm Sigplan Notices*. ACM, 2003. v. 38, n. 5, p. 1–11. ISBN 1581136625. Disponível em: <http://portal.acm.org/citation.cfm?id=781133>.

GILL, K.; YANG, S.-H. A scheme for preventing denial of service attacks on wireless sensor networks. In: *Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE*. IEEE, 2010. p. 2603–2609. ISBN 978-1-4244-4648-3. Disponível em: <http://ieeexplore.ieee-.org/xpls/abs_all.jsp?arnumber=5415233>.

HALDERMAN, J. A. et al. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM - Security in the Browser*, v. 52, p. 91–98, 2009. Disponível em: <http://dl.acm.org/citation.cfm?id=1506429>.

HART Communication Protocol and Foundation - Home Page. Disponível em: <http://www-.hartcomm.org/>. Acesso em: 26 de Agosto de 2012.

HU, F.; CHEN, J. Protecting Data Aggregation from Compromised Nodes in Wireless Sensor Network. *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, Ieee, p. 231–234, abr. 2009.

International Organization for Standardization. *ISO 13335-1:2004: Information technology – Security techniques – Management of information and communications technology security – Part 1: Concepts and models for information and communications technology security management*. [S.l.: s.n.], 2004.

International Organization for Standardization. *ISO/IEC TR 18044:2004: Information technology – Security techniques – Information security incident management*. [S.l.: s.n.], 2004. 50 p.

International Organization for Standardization. Final Draft. *ISO 27002:2005: Tecnologia da Informação – Técnicas de segurança – Código de prática para a gestão da segurança da informação*. [S.l.: s.n.], 2005. 120 p.

KUSHALNAGAR, N.; SCHUMACHER, C.; MONTENEGRO, G. IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals. p. 1–13, 2007. Disponível em: <http://tools.ietf.org/html/rfc4919.txt>.

LEVIS, P. TinyOS programming. 2006. Disponível em: <http://www.tinyos.net/tinyos-2.x-/doc/pdf/tinyos-programming.pdf>.

LEVIS, P.; LEE, N.; WELSH, M. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. *Proceedings of the 1st international*, 2003. Disponível em: <http://dl.acm.org/citation.cfm?id=958506>.

LIU PANOS KAMPANAKIS, P. N. A. *TinyECC: Elliptic Curve Cryptography for Sensor Networks*. Agosto 2012. Disponível em: <http://discovery.csc.ncsu.edu/software/TinyECC-/ver0.3/index.html>.

LOUREIRO, A. et al. Redes de sensores sem fio. In: *Simpósio Brasileiro de Redes de Computadores*. [S.l.: s.n.], 2003. v. 21, p. 19–23.

MAILING list for Avrora simulator users and developers. Disponível em: <http://lists.ucla-.edu/cgi-bin/mailman/listinfo/avrora>. Acesso em: 26 de Agosto de 2012.

MATHEWS, M. et al. Detecting compromised nodes in wireless sensor networks. *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, Ieee, p. 273–278, jul. 2007. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4287516>.

MONTENEGRO, G. et al. Transmission of IPv6 packets over IEEE 802.15. 4 networks. *Internet proposed standard ...*, p. 1–31, 2007. Disponível em: <http://www.hjp.at/doc/rfc-/rfc4944.html>.

MUNIVEL, E.; AJIT, G. Efficient Public Key Infrastructure Implementation in Wireless Sensor Networks. *Wireless Communication and Sensor ...*, Ieee, p. 1–6, jan. 2010. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5415904 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5415904>.

MURALEEDHARAN, R.; YAN, Y.; OSADCIW, L. A. Detecting Sybil attacks in image senor network using cognitive intelligence. *SANET '07*, ACM Press, New York, New York, USA, p. 59, 2007. Disponível em: <http://portal.acm.org/citation.cfm?doid=1287731.1287746>.

NEWSOME, J. et al. The sybil attack in sensor networks: analysis & defenses. p. 259–268, 2004. Disponível em: <http://portal.acm.org/citation.cfm?id=984660>.

NS-2 - The Network Simulator. [S.l.], 2012. Disponível em: <http://nsnam.isi.edu/nsnam-/index.php/Main_Page>. Acesso em: 15 de Junho de 2012.

OLIVEIRA, L. B. et al. *TinyPBC: Pairing-based Cryptography in Sensor Networks*. Agosto 2012. Disponível em: <https://sites.google.com/site/tinypbc/>.

OMNET++. [S.l.], 2012. Disponível em: <http://www.omnetpp.org/>. Acesso em: 01 de Julho de 2012.

ÖSTERLIND, F. A sensor network simulator for the Contiki OS. *SICS Research Report*, 2006. Disponível em: <http://eprints.sics.se/2296>.

POTDAR, V.; SHARIF, A.; CHANG, E. Wireless Sensor Networks: A Survey. *2009 International Conference on Advanced Information Networking and Applications Workshops*, Ieee, p. 636–641, maio 2009.

RAYMOND, D. R.; MIDKIFF, S. F. Denial-of-service in wireless sensor networks: Attacks and defenses. *Pervasive Computing, IEEE*, IEEE, v. 7, n. 1, p. 74–81, jan. 2008. ISSN 1536-1268. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4431860>.

RIZVI, S. S.; CHUNG, T. sun. *AMI: An Advanced Endurance Management Technique for Flash Memory Storage Systems*. 2008. 39–47 p.

RODRIGUEZ, H. et al. A public key encryption model for wireless sensor networks. In: *Communication Systems, Networks and Digital Signal Processing, 2008. CNSDSP 2008. 6th International Symposium on*. IEEE, 2008. p. 373–377. ISBN 978-1-4244-1875-6. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4610789>.

SOCIETY, I. C. *Local and Metropolitan Area Networks Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. [S.l.]: Telecommunications and Information Exchange Between Systems, 2003.

TINYOS - Small Operating System/Framework for Sensor Motes. Junho 2012. Disponível em: <http://code.google.com/p/tinyos-main/>.

TINYOS-BLIP. Disponível em: <http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip>. Acesso em: 26 de Agosto de 2012.

VASSEUR, J.-P.; DUNKELS, A. *Interconnecting Smart Objects with IP: The Next Internet*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN 0123751659, 9780123751652.

WALTERS, J. et al. Wireless sensor network security: A survey. *Security in distributed, grid, mobile, and*, p. 1–50, 2007.

XU, C.; GE, Y. The Public Key Encryption to Improve the Security on Wireless Sensor Networks. *2009 Second International Conference on Information and Computing Science*, Ieee, p. 11–14, maio 2009. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=5169527>.

ZHANG, Q. et al. Defending against sybil attacks in sensor networks. *Workshops, 2005. 25th*, IEEE, p. 185–191, 2005. Disponível em: <http://ieeexplore.ieee.org/iel5/9817/30953-/01437174.pdf?arnumber=1437174>.

ZHAO, Z. et al. Detecting Wormhole Attacks in Wireless Sensor Networks with Statistical Analysis. *2010 WASE International Conference on Information Engineering*, Ieee, p. 251–254, ago. 2010. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper-.htm?arnumber=5571080>.

ZIGBEE Alliance. Disponível em: <http://www.zigbee.org/>. Acesso em: 26 de Agosto de 2012.